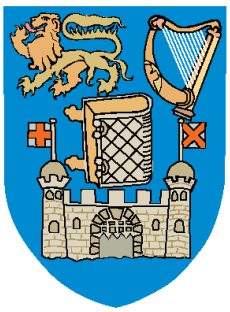
Accelerated Computing on Computational Grid Infrastructures



John Walsh BA(Mod) MSc

School of Computer Science and Statistics

Trinity College, University of Dublin

A thesis submitted for the degree of

Doctor of Philosophy (PhD)

January 2016

Declaration

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.

I agree to deposit this thesis in the University's open access institutional repository or allow the library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

John Walsh BA(Mod) MSc

Abstract

Grid infrastructures have been central to some of the largest computationally intensive scientific investigations in recent times. As the complexity of computational simulations and models increases, the dependency on faster processing power and massively parallel computing also increases. In synergy, faster processing power drives advancements in computational simulations and models. The emergence of several new types of computational resources, such as GPGPUs, Intel's Xeon Phi, FPGAs and other types of accelerators are a manifestation of this. These resources have diverse characteristics (e.g. physical properties) and there is currently no uniform way to discover or specify them in a grid job.

The diversity of the systems that make up Grids is managed by adherence to common standards that define how systems, services and users interact with Grids. While this helps provide for long-term stability, the downside is that Grids cannot handle rapid changes to the standards and grid middleware. Thus, a *Resource Integration Problem* arises, which can be summed up as a question of how to deal with the integration of new and diverse computational resources in a massive distributed grid computing environment that conversly requires stability.

The thesis shows that a conceptual approach can be used to provide a level of abstraction that assists with the process of integrating new computational resources that adhere to existing grid standards while requiring only a small number of unobtrusive changes to the grid middleware. This has never been explored before. The result is a flexible, dynamic approach to the integration of new (and yet to be conceived) resources into existing grid infrastructures. This effectiveness of this approach is demonstrated by extending the Grid to handle both GPGPU and virtual GPGPU resources.

To my parents Christopher (Christy) and Elizabeth (Betty) Walsh, my family, friends and Familien Gostic/Koroleva.

Acknowledgements

I would like to thank my supervisor Dr Jonathan Dukes for all the effort, guidance and help throughout the production of this thesis and our publications. I would like to thank Dr Brian Coghlan and Dr Gabriele Pierantoni, whose friendship, advice and help have been invaluable to me. I would also like to express my gratitude to all of my former friends and colleagues in Grid-Ireland, the Ops Centre and CAG – David, Eamonn, Geoff, John R., Kathryn, Keith, Oliver, Peter, Ronan, Soha, Stephen and Stuart. Their effort ensured that we deployed and ran a national grid service for almost a decade and pursued interesting grid architectures. Thanks to Dr Stefan Weber, Paul Duggan, Andriana Ioannou and Dr Michael Clear for helping me settle into Room 008.

My thanks to Prof.Donal O'Mahony. This work was carried out on behalf of the Telecommunications Graduate Initiative (TGI) project. TGI is funded by the Higher Education Authority (HEA) of Ireland under the Programme for Research in Third-Level Institutions (PRTLI) Cycle 5 and co-funded under the European Regional Development Fund (ERDF).

The author also acknowledges the additional support and assistance provided by the European Grid Infrastructure. For more information, please reference the EGI-InSPIRE paper (http://go.egi.eu/pdnon).

Contents

List of Figures

List of Tables

Listings

List of Algorithms

Acronyms

ALU Arithmetic and Logic Unit.

APEL Accounting Processor for Event Logs.

API Application Programming Interface.

ARC Advanced Resource Connector.

BDII Berkeley Database Information Index.

BOINC The Berkeley Open Infrastructure for Network Computing.

CDER CPU-dependent Execution Resource.

CE Compute Element.

CPU Central Processing Unit.

CREAM Computing Resource Execution And Management.

CUDA Compute Unified Device Architecture.

DIRAC Distributed Infrastructure with Remote Agent Control.

EC European Commission.

EDG European Data Grid.

EGEE Enabling Grids for E-Science.

EGI European Grid Infrastructure.

EGI-InSPIRE EGI Integrated Sustainable Pan-European Infrastructure for Researchers in Europe.

EU European Union.

FGCR Floating Generic Consumable Resource.

FPGA Field Programmable Gate Array.

GIP Grid Information Provider.

GIS Grid Information Service.

GLUE Grid Laboratory for a Uniform Environment.

GPGPU General-Purpose Computation on Graphics Processing Unit.

GPU Graphics Processing Unit.

HPC High Performance Computing.

HSA Heterogeneous System Architecture.

HSAIL HSA Intermediate Language.

HTC High Throughput Computing.

HTTP Hypertext Transfer Protocol.

JDL Job Description Language.

JSDL Job Submission Description Language.

JSON JavaScript Object Notation.

LCG LHC Computing Grid (now WLCG).

LCU Latency Compute Unit.

LDAP Lightweight Directory Access Protocol.

LDIF LDAP Data Interchange Format.

LHC Large Hadron Collider.

LRMS Local Resource Management System.

LSF Load Sharing Facility.

MIC Many Integrated Core(s).

MIMD Multiple Instruction, Multiple Data.

MISD Multiple Instruction, Single Data.

MPI Message Passing Interface.

NGI National Grid Initiative.

OGF Open Grid Forum.

OS Operating System.

OSG Open Science Grid.

PCI Peripheral Component Interconnect.

PCI-E PCI Express.

pGPGPU Physical GPGPU (i.e. not virtualised).

PS3 Sony PlayStation 3.

ReSS Resource Selection Service.

RGMA Relational Grid Monitoring Architecture.

RSL Resource Specification Language.

SE Storage Element.

SGE Sun Grid Engline.

SHOC Scalable HeterOgeneous Computing.

SIMD Single Instruction, Multiple Data.

SISD Single Instruction, Single Data.

SM Streaming Multi-Processor.

SPEC Standard Performance Evaluation Corporation.

SPMD Single Program, Multiple Data.

SQL Structured Query Language.

TCU Throughput Compute Unit.

TORQUE Terascale Open-Source Resource and Queue Manager.

UI User Interface.

UMD Unified Middleware Distribution.

UNICORE Uniform Interface to Computing Resources.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

USB Universal Serial Bus.

UUID Universal Unique Identifier.

vGPGPU Virtual-GPGPU.

VM Virtual Machine.

VMM Virtual Machine Monitor.

VO Virtual Organisation.

WLCG Worldwide LHC Computing Grid.

WMS Workload Management System.

WN Worker Node.

XML Extensible Markup Language.

xRSL Extended Resource Specification Language.

Glossary

BOINC The Berkeley Open Infrastructure for Network Computing (BOINC) is an open-source middleware that supports coordinated distributed volunteer computing applications.

CERN The Conseil Européen pour la Recherche Nucléaire (CERN) is the European Organisation for Nuclear Research. It is a world-class research facility based near Geneva on the Franco-Swiss border. Research concentrates on fundamental atomic and sub-atomic physics. It activities are funded by European member states.

CREAM A Web Service based Compute Element for the UMD/gLite grid middleware.

EGEE A European-based Grid Infrastructure (2004-2010). It was established to support distributed collaberative eSciences in Europe and beyond.

EGI A European-based Grid Infrastructure (2010-). It is has an administative and coordinating function for its NGI members, and helps shape the direction of grid-based collaboratice eScince.

gLite a lightweight grid middleware developed for the LCG/WLCG and EGEE/EGI.

Globus The Globus Toolkit is an standards-based open-source suite for building grids.

Grid-Ireland An Irish Goverenment funded and supported project to developed a National Grid Initiatve and Grid Operations Centre for Ireland (1999-2012).

SPEC Standard Performance Evaluation Corporation.

VO A Virtual Organisation (VO) is a managed collection of distributed users sharing a common e-Infrastructure..

XML The Extensible Markup Language is a structured software- and hardeware-independent data format used to store and transport data.

Chapter 1

Introduction

Grid computing [1] is a standards-based computing architecture that facilitates controlled, secure sharing of computational and related resources (e.g. storage and application software) among many users distributed across the Internet. The motivating factors driving Grids forward include: the ability to increase the scale and variety of resources available to users and their applications; improved resource utilisation by the smart distribution of application workloads (jobs) throughout the Grid – selection is determined based upon user specified requirements; and to allow distributed collections of collaborating users to securely share access to their common data. Grids have been successfully used by tens of thousands of users (predominantly research scientists), who have used them to investigate problems that are intractable on isolated, single-site resources. The most notable and successful scientific outcome involving Grids was the confirmation of the existence of the Higgs Boson – an endeavour that led to the joint awarding of a Nobel Prize in Physics to Professor Peter Higgs in October 2013, some 50 years after his original theory was proposed [2]. The Grid was used to process and analyse the 15 petabytes of data produced annually by the LHC experiments [3].

At its conception, grid computing had focussed on a "single program/single CPU" execution model, supporting only sequential, geometrically parallel or other ensemble algorithms (e.g. parametric sweep). For the past decade, however, the exponential growth of CPU speed and processing power has plateaued [4] [5], and this has generated many questions about the future of computational-based scientific research using this model. Support for more generalised CPU-based parallel execution frameworks that can help alleviate the slowdown of growth in performance, such as OpenMP [6] and the Message Passing Interface (MPI) [7], has become commonplace in grid in-

frastructures [8]. However, the emergence of both many-core and massively-parallel computational accelerators, for example General-Purpose Computation on Graphics Processing Units (GPGPUs), has led to users seeking access to these resources on grid infrastructures [9].

This thesis is concerned with the problem of *integrating* many new types of computational accelerators into Grids. The term "integration" is used in this context to refer to discovering those new resources, and selecting them for workload execution in a manner that grid users are already familiar with. The integration problem arises because Grid infrastructures, which require operational stability, are adverse to the rapid changes that the integration of many new types of resources require.

1.1 Motivations

The integration of massively-parallel computing resources into Grids, such as the European Grid Infrastructure (EGI), has thus far been 'ad-hoc' in nature, and no community-accepted solution has been forthcoming. Furthermore, the integration attempts that have been made to date overlook some very important advantages of Grid Computing – namely, the ability to "discover" these new resources based on their particular properties (e.g. memory, capacity, and utilisation), and the ability to specify those properties in the user's grid applications in the same way that users currently specify Grid resources based on specific CPU properties or utilisation characteristics. It should be noted that use of the term "Grid" in this thesis refers to particular coordinated collections of distributed computing resources that use the Open Grid Forum (OGF) Grid Laboratory for a Uniform Environment (GLUE) standards to publish information about the state of their resources and services.

The issue of GPGPU integration into Grids is now such an important topic that the EGI, which operates one of the largest grid infrastructures worldwide, started to actively pursue this research question in the Summer of 2015 through the EGI-Engage project [10]. However, the proposed scope is too implementation specific (i.e. it is focussed solely on particular GPGPUs), and does not look at the more generalised problem of integrating other types of accelerators. The lack of a generalised approach that offers grid discovery and user support for accelerators presents a challenge to the integration of a diverse range of computational resources in existing grid infrastructures. There is an opportunity to examine if a generalised approach to resource integration

can be developed.

In addition to the demand for access to new resources, the emergence of the Cloud Computing paradigm over the last decade is also influencing the provision of resources in grid infrastructures [11]. Cloud Computing is aimed at delivering greater flexibility, utilisation and security in how computational resources are provided and used. Virtualisation is the fundamental technology underlying Cloud Computing. In some cases, virtualisation can enable a degree of resource "mobility" – i.e. the resource can be transparently accessed from another remote machine. Resource mobility has the potential to be exploited in a number of distinct ways: (i) creating computing environments with dynamically allocated hardware; (ii) using dynamic allocation to improve resource utilisation; and (iii) constructing complex machine environments from these virtual resources that may not have been possible with conventional physical hardware. There has been a shift within the Grid community towards a more Cloud-like delivery of computing services to users. In the case of EGI, a new infrastructure that runs alongside the existing Grid was established. Neither of these infrastructures consider the potential benefits and application of resource mobility.

There are several motivating factors behind the work presented in this thesis, some are based on fulfilling immediate needs, some are based on having to work within the limitations of slowly changing infrastructures, while others include the potential for improved resource allocation and performance. These motivations include:

(i) the need to incorporate the use of massively-parallel computing resources. These resources are needed to support innovation in the sciences.

(ii) the need to support these resources in a manner that current users are accustomed to;

(iii) the need to avoid radical changes to the grid (infrastructure and middleware);

(iv) the opportunity to develop unconventional computer systems that simplify application development; and

(v) the opportunity to investigate if Grids can exploit virtualisation and virtual resources, and whether these will improve resource utilisation and allocation – thereby increasing the throughput of work through the Grid.

Personal Perspective

I have been professionally involved in the Grid community and the provision of grid services and infrastructures since mid-2003. Since then, I have seen Grid Computing growing from its experimental beginnings (delivering computational power to a small number of both large and small scientific communities), to the creation of the EGI, a massive, but coordinated, pan-European infrastructure. As one of the Irish representatives in a sequence of successive co-funded European grid projects, I also played a small role in the EGI's development and in the development of grid software (also called grid middleware).

One characteristic of huge infrastructures is that changes to them are slow and incremental – the investment made in such infrastructures (e.g. financial, software and infrastructural development) requires long-term planning in order to offer sustained stability and availability. In addition, any changes or new developments will also require input from key stakeholders (e.g. users, resource providers, software developers and testers). One of the negative side-effects that this long-term development cycle has on the grid community can be seen in how long it takes to integrate new components or services. Indeed, my experience of being involved with the development and integration of a grid-based solution for parallel computing applications using MPI was that the process took many years before a full solution was implemented. Similarly, when Grid-Ireland (p.65), the operators of the Irish National Grid Initiative (NGI), attempted to integrate the Sony Playstation 3 – a low-cost accelerator-based "supercomputing" resource – into its grid infrastructure, problems in describing and advertising the capabilities of the new resources were encountered. The same problem reoccurred when Grid-Ireland sought to integrate GPGPU resources. In both cases, the pragmatic, but incomplete, solution was to simply restrict the resources to a particular set of grid users. These users needed to follow specific instructions detailing how to access the GPGPUs. The solution was unsatisfactory, as there was no means for the users to determine if the resources were over- or under-utilised.

1.2 Objectives and Research Question

The core objective set out in this thesis is to conceptualise, investigate and develop a flexible, dynamic approach to the integration of new (and yet to be conceived) resources into existing grid infrastructures. On the basis of developing such an approach, it will be further investigated to see if it can be extended to virtual resources, and whether the use of these virtual resources can improve resource utilisation on the Grid.

1.3 Scientific Contribution of the Work

The contributions of the work presented can be categorised into two distinct areas: (i) the development of an abstract approach to the integration of accelerator-based resources into Grids (Chapter 3); and (ii) an extension of this abstraction to virtual resources, exemplified by the development new services supporting the integration of virtual-GPGPU computational resources into Grids (Chapter 4). A comprehensive list of individual contributions are detailed below.

Abstract Approach to Resource Integration

(i) The development of an abstract classification of non-CPU computational resources – these will be referred to as *CPU-Dependent Execution Resources (CDERs)*. This definition is intended to exclude applications or software (these are already facilitated in Grids) but include hardware resources such as GPGPUs, Field Programmable Gate Array (FPGAs) and other hardware accelerators. The definition is also intended to exclude software that may be "node-locked" due to licensing restrictions – these play no role in the actual execution of computational instructions.

(ii) An investigation into a set of Conceptual Strategies that aid the description and advertisement of CDERs within a Grid that is consistent with community-accepted standards.

(iii) The development of a *two-phase* process that facilitates the discovery of the CDERs and the selection of grid providers according to some user-specified CDER requirements.

(iv) The development of a new solution that allows GPGPU resources to be incorporated into a popular resource management system – GPGPUs are not currently well supported by this resource management system. The solution is extended to Grids by way of the previously mentioned investigation.

Abstract Extension to Virtual Computational Resources

(v) The design of an abstract architecture for managing a class of virtual-CDER – the virtual GPGPU.

(vi) A concrete realisation of this abstract architecture, which is applied for use by several (more general) computer systems.

(vii) The extension of the abstract virtual-GPGPU model to a grid context.

(viii) A concrete realisation of the abstract virtual-GPGPU grid model that incorporates the two-phase model.

1.4 Related Peer-Reviewed Publications

The body of work presented in this thesis is based on a series of peer reviewed publications.

- **Supporting grid-enabled GPU workloads using rCUDA and StratusLab** [12]
  This preliminary work presented a very early conception that featured both GPGPUs and virtual-GPGPU resources on a grid infrastructure. The publication included an initial study of the challenges that the grid community would need to tackle in order to make GPGPU resources available on the Grid. Furthermore it presented and demonstrated a simple application that transparently executed GPGPU kernels (i.e. instructions that execute on GPGPUs) on several remote virtual-GPGPUs as if they were locally installed. Finally, a system architecture that supported virtual-GPGPUs was proposed.

- **Overview and Evaluation of Conceptual Strategies for Accessing CPU-Dependent Execution Resources in Grid Infrastructures** [13]
  Five conceptual strategies for discovering and accessing CDERs were described and evaluated against key criteria. All five strategies were compliant with current standards. From this evaluation, two of the strategies clearly emerged as providing the

greatest flexibility for publishing both static and dynamic CDER information and identifying CDERs that satisfy specific job requirements. Furthermore, a two-phase approach to job-submission was developed for jobs requiring access to CDERs.

- **Supporting job-level secure access to GPGPU resources on existing grid infrastructures** [14]
  This publication introduced an update to grid infrastructures using the gLite [15] grid middleware. The update supported GPGPU resource discovery and job specification on the grid. The publication presented a technical discussion on the implementation details behind the application of a conceptual strategy [13] to GPGPU resources. A new service was developed to help manage the allocation of GPGPU resources and ensure additional GPGPU resource access protections that had not been available in the popular TORQUE/MAUI resource allocation system. This system is the most popular resource management system deployed on the Grid (see Section 2.4.4).

- **GPGPU Virtualisation with Multi-API Support using Containers** [16]
  This publication described a multi-layered approach to GPGPU virtualisation where user applications could transparently access many non-local GPGPU resources. A new multi-component system that supports multiple remote virtual-GPGPU implementations on several resource management systems was proposed. The system consisted of: (a) a Factory component that produces Virtual Machines with access to a unique single GPGPU; (b) a Registry service that managed allocation of the Factory produced resources; and, (c) a set of plug-in scripts that bridge between the resource management system and the Registry. This paper evaluated the performance of GPGPU benchmarks on the prototype.

- **Application Support for Virtual GPGPUs in Grid Infrastructures** [17]
  This publication built upon the results of the previous publications ([13], [14], [16]), to extend grids with support for virtual-GPGPU resources. The support included virtual-GPGPU resource discovery, job description, and selection. Performance results suggested that improved resource utilisation could compensate for the overhead of access to the virtual-GPGPUs.

- **Many-core on the Grid: From exploration to production** [18][1].
  This paper discusses the key challenges of supporting many-core devices and applications on the LHC Computing Grid Infrastructure.

1.5 Structure of the Thesis

Chapter 2 – Towards Accelerated Computing on Grids

Chapter 2 presents a layered view of the background and state of the art of grid computing using computational accelerators.

Section 2.1 introduces the fundamental concepts and terminology behind Cluster Computing and cluster resource management. As efficient resource management is a crucial aspect of how clusters function, an overview of the state of the art is presented, illustrating the differences between the numerous resource management systems in use today. Support for non-CPU resources, such as physically-bound GPGPUs or machine-independent resources (software licences, virtual resources) is also reviewed.

Section 2.2 begins with a description of the basics of accelerated computing, and the reasons why it has become important to contemporary scientific computing. With much focus on accelerating applications using GPGPUs, a basic overview of GPGPU parallel computing models is presented. The different application development and runtime frameworks (APIs), hardware and examples of user applications are also reviewed.

Section 2.3 explores topics in machine and resource virtualisation. Machine virtualisation has led to the rapid development of computing paradigms such as Cloud Computing, which has transformed how computing resources are shared and used. This section also looks into the fundamentals of virtualising the GPGPU. Machine and resource virtualisation is utilised in Chapter 4 to support the integration of virtual-GPGPUs into the Grid. One goal of GPGPU virtualisation in this thesis is to improve GPGPU resource utilisation.

Grids are comprised of many clusters distributed across the Internet. The fundamentals of Grid Computing are introduced in Section 2.4, describing how a coordinated ensemble of distributed resources can be used to tackle problems that are otherwise infeasible at a local level. A basic introduction to resource discovery and job submission is introduced and will be used to illustrate the differences between some grid systems. These differences are exemplified in how the various grids support job submission with automatic resource selection. Section 2.4 also discusses how current grids lack the flexibility to support accelerated computing, and sets out how this work contributes towards alleviating this problem. In addition, Section 2.4 surveys current trends in

[1]Non-primary contributing author

accelerator deployment and usage in computing environments. It also examines the deployment of grid midddleware and their resource management systems. This is used to evaluate which grid systems currently support accelerator integration at the resource management level, and the extent to which the solutions presented in Chapter 3 and Chapter 4 can apply. This section will also review where previous attempts have been made to integrate accelerators, and how the solutions presented in this thesis overcome problems associated with those solutions. Finally, the responses to two surveys are analysed to gather further information about accelerator deployment and how grid users intend to use them.

Section 2.5 examines other research that has similar goals (albeit, focussed on more narrow problems) to the accelerator integration problem. A summary of the differences between the work presented in this thesis and the related work will be provided.

Chapter 3 – CPU-Dependent Execution Resources in Grid Infrastructures

Chapter 3 is a greatly expanded and updated treatment of the work presented in the "Overview and Evaluation of Conceptual Strategies for Accessing CPU-Dependent Execution Resources in Grid Infrastructures" [13] and "Supporting job-level secure access to GPGPU resources on existing grid infrastructures" [14] papers. It evaluates the current grid information model, which is a core component in how grid resources are discovered and selected for use. This chapter introduces *CPU-Dependent Execution Resources* (CDERs) – an abstraction of a set of resources that require the allocation of one or more CPU cores in order to gain access to the resource itself. The evaluation primarily focusses on approaches that may be used to extend the information systems. An example of how CDERs, in the form of a General Purpose Graphics Processing Unit (GPGPU), can be applied and integrated into a Grid System is presented in Section 3.3. This section not only covers how the Grid Information System is extended, but also what other middleware extensions are made to a grid infrastructure to make these resources accessible. The approach taken is intended to be lightweight, flexible and extensible – the same method can be used to facilitate other type of CDERs, such as Field Programmable Gate Arrays.

Chapter 4 – Grid-enabled Virtual GPGPUs

Similar to the approach in Chapter 3, Chapter 4 is based on a merged and expanded treatment of two peer-reviewed papers: "GPGPU Virtualisation with Multi-API Support using Containers" [16] and "Application Support for Virtual GPGPUs in Grid Infrastructures" [17]. A derivation of the *CPU-Dependent Execution Resources* (CDER) concept is developed that allows the inclusion of non-CPU computing resources that can be exclusively accessed from *any* machine in a cluster. These resources still require one or more CPUs to access them. In addition, because the resources can be accessed from any machine, they can be regarded as *mobile* or *floating* resources[1]. A motivation for developing this derivation is to facilitate the management of floating virtualised resources, in particular a new class of virtual-GPGPUs resources.

Section 4.1 examines how an abstract multi-layered approach to GPGPU virtualisation may be used to describe an architecture that can support virtual-GPGPU resources across many cluster systems. It is then shown in Section 4.2 how to apply this abstraction to a couple of contemporary resource management systems. The implementation is supported through a set of new components that add a new lightweight system for constructing, managing and integrating these virtual-GPGPUs.

The ability to support virtual-GPGPUs with multiple cluster resource management systems is central to its potential for adaptation on grid infrastructures – Grids are managed collections of diverse clusters. Section 4.3 proposes how the abstract cluster model can be extended to Grids, while Section 4.3.3 presents an implementation.

The speed and throughput of virtual-GPGPU applications is studied in Section 4.4 to determine under what conditions their virtual-GPGPUs are effective.

Chapter 5 – Conclusions and Future Prospects

This final chapter recalls the research question of how to deal with the integration of new computational resources in a massive distributed grid computing environment that (somewhat conversely) requires stability. A summary of how an abstract approach (one of the key contributions) to this resource integration question recaps how this problem is addressed – first, in the context of (physical) GPGPU resources, and then in the context of an extension to virtual-GPGPU resources.

The limitations of the research, and the potential for further research to ameliorate

[1]The standard terminology is "floating resource"

those limitations, is addressed. For example, through the development of additional resource management services, the question of how to expand the new services to deal with other resources is discussed, as well as the potential of investigating alternative resource allocation schemes that may increase utilisation.

Finally, some ideas on potential new research topics that enhance the discovery, selection and integration of both new and older resources shall be discussed. These topics touch on technologies which are not currently used in the grid ecosystem (but are used elsewhere), and were beyond the scope of this thesis.

Chapter 2

Towards Accelerated Computing on Grid Infrastructures

Computational- and data-based sciences (the so-called e-Sciences) are becoming more reliant on parallel processing to deal with increasingly complex simulations and data-sets. Such parallel processing applications are typically executed on multiple CPUs across a network of computers, or executed on large multi-processor computers. However, with the slowdown in the growth of processor performance, alternative solutions that use additional specialised computational resources to supplement the computational power of the CPU have also emerged – namely *coprocessors* and *computational accelerators*. One of the big challenges for grid infrastructures is that these resources are neither easily nor quickly integrated into grids in such a way that it is supported by typical higher-level grid functions (e.g. resource discovery and the ability to specify the need for that resource in a grid job). Indeed, based on previous experience in developing a standard for parallel application support using MPI on European grids (under the auspices of the EGEE MPI-Working Group), it took several years before our initial recommendations were fully supported in the grid middleware (e.g. WholeNodes support that allowed grid jobs to request all CPUs of a machine).

This thesis is principally concerned with developing a flexible way to integrate new computational resources into grid infrastructures. One of the core hypotheses is that a conceptual approach to this resource integration problem can lead to a more generalised solution that is quicker, more flexible (allows many resource types) and more powerful (supports discovery and resource matching) than the approaches that have been used to date. The method developed is applied to an important class of resources that are able

to process large quantities of calculations in parallel – the many-core accelerator based on general purpose computations on graphics processing units. However, the general method is intended to be able to support other resources, such as Field Programmable Gate Arrays.

The approach taken in this chapter is to present the background material in a layered, bottom-up fashion. Grids are built upon collections of distributed *clusters* of computing resources, so Section 2.1, which introduces the basics of cluster-based computing, is a natural starting point. This introduction sets forth key concepts and terminology. These concepts are also important in understanding the low-level details of how Grids work. In particular, knowledge of basic cluster computing concepts is required for the work presented in Chapters 3 and 4.

Section 2.2 looks at the topic of accelerated computing. This examines the transition from the classic CPU-based serial computing architecture (i.e. only one computer program instruction can be executed at any instant) to parallel-based computing architectures (where several, or indeed many, instructions can be executed concurrently). An overview of the subsequent serial and parallel computing architectures is presented. Furthermore, as there are several types of CPU and accelerator hardware, with each of these implementing a different approach to parallel computing, a review of some common CPU and accelerator implementations, and how these relate to the aforementioned parallel computing architectures, is also presented. The remainder of this section investigates different application interfaces that are required to access accelerators, and how they are currently supported by several cluster resource management systems.

An overview of current methods for benchmarking accelerators is presented in Section 2.2.5. Benchmarking has two major functions in this thesis: (i) it provides key information about resource capabilties in a grid (which in turn enhance resource selection); and (ii) it helps evaluate the impact of virtualising GPGPUs. This section investigates some common accelerator benchmarking suites, looking at several individual benchmarking applications and what key performance measurements they make.

Machine and GPGPU virtualisation is introduced in Section 2.3. The primary benefit of virtualisation is that it can provide better flexibility in sharing and allocating limited resources, and in doing so, increase a resource's utilisation. Virtualisation is now a central pillar of contemporary computational service provision. The grid resource integration method advocated in Chapter 3 is adapted to implement a Virtual GPGPU grid service in Chapter 4.

The fundamentals of grid computing are described in Section 2.4, with particular attention being paid to job submission and the information model. This section also reviews previous work that has attempted to introduce additional computational (non-accelerator) resources into the grid. Finally, an analysis of key results from the *EGI Resource Centre and Users* survey is presented. This examined whether there was a need for GPGPU resources on the grid, and what those needs were. A grid integration solution that takes the survey results into account is presented in Chapter 3 and developed further in Chapter 4. This is followed by a survey of the actual composition of the EGI via its information system.

Finally, Section 2.5 looks at the state of the art and related work to examine what previous and current works attempt to solve similar problems to the work proposed in this thesis. Where similar work exists, it shall be compared to the proposed solution.

2.1 Cluster Computing

A *Cluster* is the general term used to describe a collection of locally inter-connected computers (i.e. all computers are in the same physical location) that act together to complete large-scale computational tasks. The local inter-connection between the computers ensures that when two or more computers are communicating with one another during the execution of a common task, the amount of time that it takes to send data from one computer to another is minimised, and that the impact on communication by events external to the local network (e.g. network outages) is minimised. Each computer in the cluster that participates in the processing of computational tasks is known as a Worker Node (WN).

The operational cost (e.g. expert staff, electricity, maintenance, equipment depreciation) of running a cluster is high. To reduce these costs, and to achieve the best return on investment, it is important to ensure that the resources are used as much as possible. One way to ensure greater utilisation is to share the cluster resources between many *users*. Indeed in some instances, clusters may have hundreds or thousands of users. To effectively share the cluster between those users, some form of resource (e.g. CPU cores, memory) management is required. Such a system is called a Local Resource Management System (LRMS) or *Batch System*. There are many LRMS application suites available, including TORQUE [19], SLURM [20], LSF [21], GridEngine [22], Condor [23] and OAR [24].

The LRMS service runs on a *Head Node*, whose function is to allocate the worker node resources in such a way that the users can complete their task with the desired number of resources. By managing when each users application executes, the LRMS ensures that user applications do not concurrently compete for the same resources, thereby ensuring that there is no over-commitment of individual resources. Furthermore, the LRMS may be configured to *schedule* tasks according to some predefined policy. Preference may, for example, be given to tasks that require the smallest amount of resources, or may be given to certain groups of users. Some LRMS's allow the scheduling function to be handled by an external manager. For example, MAUI [25], and its commercial successor MOAB [26], implement advanced scheduling policies such as *Fair Shares*, that ensure certain groups of users have a guarantee of a fixed amount of resources over a fixed period of time.

Collections of worker nodes may be arranged into groups (*Queues* or *Partitions*). Queues are just one of several ways to manage access policies or set limits on how users access the worker node resources. For example, Queues can limit which users can access particular worker nodes, how long their job may execute for, or how much memory they may use.

The user's workload is called a *Job*. Each Job is specified by using a *Job Description* or *Job Specification*. The job description should include *Batch-System Directives* which state the job's resource requirements. Jobs are submitted from a *Job Submission Node* – in many cases the Head Node and Job Submission Node run services on the same machine.

A typical job specification may include a directive for how many Worker Nodes are required (a *Node* or *Task*). In most batch systems, this actually translates into a request for the number of CPU cores (or Processors), and those cores may be allocated on a single worker node, or spread out across multiple worker nodes. However, the specification can be refined to ensure that a set of CPU cores are on the same worker node. This is called *Processors Per Node*.

Each job requires one or more CPU cores to execute. For this reason, the CPU core is the primary resource managed by the LRMS. However, users may be interested in using other types of resources – for example GPGPUs or software licences. In the first case, the resource is physically attached to a worker node, whilst in the latter case, the resource may be accessed from any worker node. These are respectively known as *consumable resources* and *floating consumable resources*.

Cluster systems can range from very small collections of worker nodes to huge systems with over 100,000 CPUs that are interconnected through high-speed components. Some of these more important differences can be classified into the following computing models:

High Performance Computing (HPC) uses computer components that are focussed on delivering the optimal computational power and capability possible, thereby allowing user jobs to execute as quickly as possible. HPC jobs and systems are normally *tightly-coupled*, i.e. there is significant inter-dependence between different parts of the application executing on the system (e.g. shared memory). HPC systems may be shared among hundreds or thousands of users from different scientific backgrounds and with different application needs. HPC systems require an LRMS to optimise utilisation without overcommitting resources.

High Throughput Computing (HTC) places less emphasis on execution speed, but is focussed mainly on the execution of many loosely-coupled (the converse of tightly coupled) user jobs over an extended period of time. Collections of user jobs can be be executed over months and years, rather than hours and days (e.g. LHC [3]).

Although HPC systems are primarily intended for large multi-worker node applications that depend upon fast inter-node communication, they have also been used for large collections of single processor applications. Similarly, HTC systems are primarily aimed at processing large numbers of single processor or small multi-core jobs, but have also been used for multi-processor applications. Indeed, some cluster systems are intended to manage a diverse range of applications, so they must embrace elements of HPC and HTC system architectures. The inclusion of accelerators on worker nodes is just one of the ways to achieve this.

2.2 Accelerated Computing

A coprocessor is a hardware component that supplements the capabilities of the primary processor (CPU). The use of additional hardware to boost or supplement the capabilities of the CPU has been around for several decades. For example, the Motorola MC68882 floating-point unit was an early coprocessor that specialised in performing

mathematical operations at a time when the typical CPU could only handle floating-point calculations in software. Such coprocessors provided a much needed boost to application performance, but were generally fixed-purpose and limited in their function. Moreover, access to the coprocessor wasn't always transparent, as it often had to be accessed using an additional API. Some coprocessor functions, such as the floating-point unit, are now integrated directly into the CPU logic, and consequently do not need to be provided by an external coprocessor.

It is clear from the above example that a coprocessor may have a fixed function (e.g operate on floating-point numbers). For accelerated general-purpose processing a more precise definition is required. In this thesis, Volodymy Kindratenko's definition of a *Computational Accelerator* is adopted. He defines a computational accelerator to be "a reprogrammable, stand-alone processor capable of executing an arbitrarily complex, but functionally limited code, offering the benefit of faster execution as compared to the general-purpose processor. The functional limitation comes from its inability to directly interact with some of the hardware resources that are typically accessible by the main processor, such as disk or network interface. It is not a fixed hardware designed to accelerate the execution of a particular instruction (e.g., a vector unit) or a particular fixed function (e.g., H.264 media decoding)" [27]. However, in the body of this thesis, the shorter term of *accelerator* may be used, with the understanding that this refers to a computational accelerator.

The crucial points to be taken from this definition is that accelerators are *separate* from the CPU, but are dependent on the CPU to interact with the rest of the machine. Furthermore, accelerators will have their own set of states and properties that are distinguishable from the CPU. It is this separation property that has the effect that accelerators (and coprocessors) are not catered for in contemporary grid infrastructures.

The term *Heterogeneous Computing* is used to describe a system architecture that is composed of several different processing units. Such systems will include one or more (not necessarily identical) CPUs, but may also include one or more general purpose accelerators, as well as digital signal processors or other fixed purpose coprocessors.

2.2.1 From Single- to Multi-Core and Manycore Computing

Since it was first observed in 1965, the rate at which newer manufacturing processes allowed the size of transistors in electronic circuits to be halved was approximately every

18 months. A side effect of this miniaturisation was that the rate at which processors can execute instructions also doubled. This rule-of-thumb is known as Moore's law. However, since 2005 that growth rate has effectively plateaued because of the physical difficulties in manufacturing at nano-scale. While instruction processing techniques such as super-scaling, instruction pipe-lining and caching [28] have helped improve performance, these have not been sufficient in themselves to maintain expected performance increases. The use of parallel computing techniques is regarded as one of the best available ways to overcome some of these limitations.

Parallelisation is by no means a recent concept and many examples of its use can be found in the natural world (e.g. independent worker bees in a hive), as well as in manufacturing processes such as assembly lines (that are based around complex workflows that depend on synchronisation in order to efficiently assemble the intermediate- and end-products). In computer science, one of the first taxonomies of computing architectures that describes how serial and parallel processing may be approached has come to be known as Flynn's Taxonomy [29] [30]. This divides computational architectures into one of four types:

SISD Single Instruction, Single Data – programs execute sequentially: One instruction operates on one piece of data, and then moves onto the next. This has no parallel computing capabilities.

SIMD Single Instruction, Multiple data – in which the same instruction is applied to a set of data in parallel. A well-known example of this architecture is vector processing.

MISD Multiple Instruction, Single Data – this is deemed to be a less likely computational scenario. However, it has uses in fault-tolerant systems [31], where the redundancy of executing the same instruction in parallel on the same data can be used to ensure better confidence that an error has not occurred, or can be detected (and eliminated) through checksumming or consensus.

MIMD Multiple Instruction, Multiple Data – this is the broadest parallel computing model, and allows different streams of instructions to be executed in parallel on different data. The MIMD model encompasses models that facilitate collections of SISD processors (e.g. multicore CPUs) and collections of SIMD processors (e.g. manycore GPGPUs).

As the MIMD architecture is very broad in its classification, further refinements of MIMD have been adopted. This helps distinguish between significantly different MIMD architectures [31]. The extended Flynn's taxonomy defines the following additional categories for parallel processing:

Shared Memory MIMD All the memory is shared between each processing unit. This category includes CPUs with multiple processing cores on a single processor, and machines with multiple processors which may also have multiple processing cores. When all processing cores are equivalent, this form of Shared Memory architecture is called a *Symmetric Multi-Processor* (SMP) model. An alternate shared memory MIMD is the master/worker model where some of the processors are assigned to execute specialised workloads or software. This model includes accelerators (including fixed function accelerators such as H.264 decoders).

Distributed Memory MIMD This includes processors that communicate through message passing. Memory is not directly shared by any of the processors, and data is shared by sending messages from one processor to another. This is also referred to as a Single Program, Multiple Data (SPMD) architecture, and has been popularised on Clusters through the popular MPI application development interface.

The use of these classifications is dependent on the specific context (granularity) in which a computer architecture is set. So for example, if a program executes on a uniprocessor machine, then it follows an SISD architecture, but when it is part of a set of distributed processes executing on a cluster it operates in an SPMD environment. SIMD and MIMD architectures have always been at the forefront of performance computing. With the slowdown in SISD performance growth, computing based on both SIMD and MIMD architectures are now key to satisfying the computational processing needs for eScience, industry and users alike.

Processor and Accelerator Architectures

To illustrate the basic differences between architectures, several processors and accelerators are considered. Firstly, there are significant architectural differences between one type of processor and another, and between one type of accelerator and another; secondly, as will be later shown, the distinction between a set of processors and some

coprocessors/accelerators can seem to be somewhat blurred (e.g Intel Xeon Phi in *native* mode). The LRMS that manages the worker node resources will have to handle whether such accelerators are treated as a large multi-core worker node or as an accelerator.

Single-core CPUs

The most basic of these processor architectures is the uniprocessor (single core processor). This implements the SISD architecture, by executing a single instruction on its data at a time. The single core processor is also a fundamental resource managed by an LRMS. To execute parallel code, unicore processors either have to support vector instructions, or use message passing across a network (e.g. MPI).

Multi-core CPUs

Multi-core CPUs allow several independent processing cores on the same CPU to act independently of each other by executing their own SISD instruction streams. As illistrated in Figure 2.1, each core acts like a uniprocessor and has its own control-unit, registers, and arithmetic and logic unit. Fast access to data is managed through the core's local cache, and a higher-level shared memory cache ensures that data can be consistently shared between two more cores.

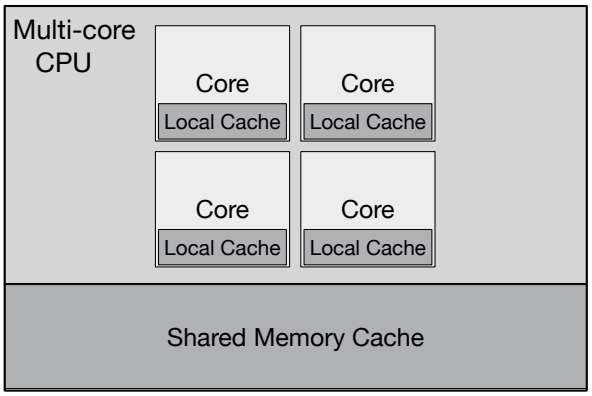


Figure 2.1: A simplified Multi-core CPU with 4 Cores.

Some multi-core CPUs allow individual cores to operate at different speeds, where each core can adapt to the processing demands placed on it. This allows the CPU to operate more power efficiently. Furthermore, as each core acts independently, they can be represented in the LRMS as atomically allocatable units. However, a user submitting a multi-core job into an LRMS may be interested in ensuring that two or

more identical cores are physically on the same machine (or even processor). This is often referred to as the SMP size or SMP granularity. Multi-core machines can be used as the basis of Distributed MIMD and Shared MIMD architectures.

Manycore Accelerators – Xeon Phi

Intel's Many Integrated Core (MIC) Xeon Phi [32] was introduced in 2012 as a peripheral device that can be added to an existing physical machine (the host). It is an accelerator composed of 61 modified Intel Pentium processors, which are inter-connected through a high-throughput bi-directional communication ring. The processors can execute x86-compatible applications, and they share "on-accelerator" global memory. This memory is not available for direct use by the host. Furthermore, each core supports four simultaneous threads and it shares its cache memory between those threads. In addition, each core supports 512-bit SIMD vector processing. A simplified overview of the Intel Xeon Phi (Knight's Corner) is illustrated in Figure 2.2, where only a small sample of the all the components is included.

The Xeon Phi can operate as a networked host using an embedded Linux operating system (leveraging the physical machine). This host operates like a large SMP machine in its own right (users can log on and execute x86-compatible applications). This is known as *native* mode. In this mode, the Xeon Phi can act as worker node in a cluster. *Symmetric* mode uses both the physical host and the Xeon Phi's native mode to execute applications across both sets of processor resources. Finally, in *offload* mode, regions of application code can be offloaded to execute on the Xeon Phi using either *Compiler Assisted Offload* (i.e. implicit or explicit compiler directives) or at runtime using *Automatic Offload*. In Automatic Offload the Intel Math Kernel Library manages when regions of the application are executed on the host or on the Xeon Phi.

The native mode and symmetric mode usage does not quite fit into Kindratenko's definition of an accelerator – in both cases the Xeon Phi can be accessed like a normal node. From a grid integration perspective there is no challenge – these nodes can operate together as a standard cluster on a grid. However, if the Xeon Phi "nodes" are not directly integrated into the cluster (i.e. the LRMS does not manage their allocation/deallocation) or if access is through Offload mode, then there is a grid integration challenge. Namely resource advertisement, discovery, and job description must somehow be supported.

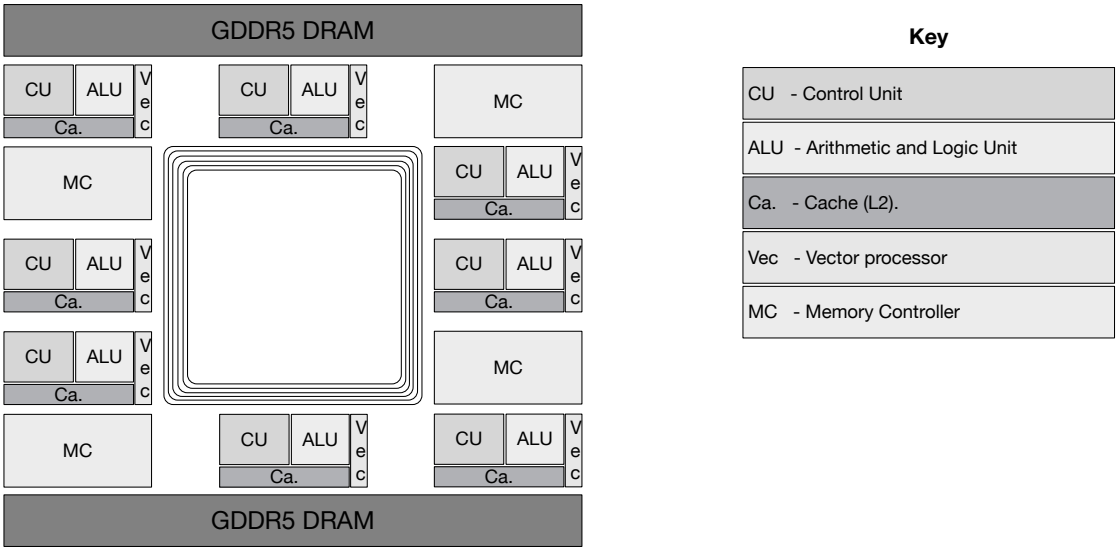


Figure 2.2: The Intel Xeon Phi (Knight's Corner) Architecture. This includes 61 enhanced Pentium processors, each capable of executing 4 threads and performing vector operations. The processors are interconnected on a 6-lane data and instruction bus. Advanced shared global memory management is supported through a set of inter-connected memory controllers (MC)

Manycore Accelerators – Nvidia GPGPUs

The Nvidia GPGPU is one of the most successful and enduring accelerators on the market. It reuses and extends hardware originally designed and intended for graphics rendering, and applies it to more general numeric computational processing. The success of GPGPUs is partly due to their low cost – a basic Nvidia GPGPU can be purchased for less than fifty euro, but more advanced models may cost several thousand euro.

A fundamental processing unit in a GPGPU is the *Streaming Multi-Processor* (SM). The number of streaming multi-processors varies from one GPGPU model to the next. Each of these Streaming Multi Processors have their own local cache and control units, as well as a large number of smaller arithmetic and logic cores (Streaming Processors) that perform the numerical calculations. Each SM conforms to the SIMD architecture. A simplified overview of the GPGPU's architecture is depicted in Figure 2.3, where each of the Streaming Multi-processor ALU cores executes the same instruction on its own data.

Unlike the Xeon Phi, the Nvidia GPGPU does not have a system-on-chip mode of operation, and cannot be accessed without using the host machine. The host is required for moving data to and from the GPGPU and for queuing GPGPU commands.

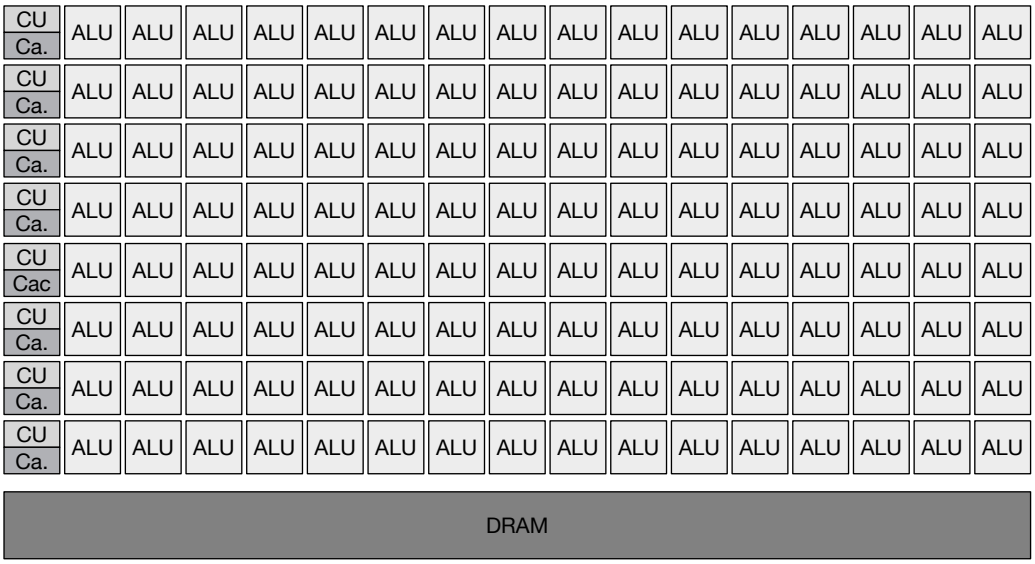


Figure 2.3: The Many-Core Nvidia GPGPU Architecture

Heterogeneous System Architecture

The Heterogeneous System Architecture (HSA) is a multi-vendor initiative that seeks to transparently amalgamate the processing power of distinct processing units (e.g. CPUs and GPUs) on the same physical machine.

Normally, the CPU and the accelerator on a non-HSA system have distinct memory management. This is particularly true when the accelerator is an attached device. When the accelerator needs to process a large set of data, the data must first be copied from the host's memory to that of the accelerator. Even when the the CPU and accelerator share the same physical memory (e.g. the GPU is incorporated into the CPU), the memory will be partitioned between the host and the device, so the data copying operation is still required. The HSA architecture defines a unified virtual address space, that helps avoid the need to copy the data.

Another notable feature of the HSA architecture is that it defines two types of generalised processing units: (i) The Latency Compute Unit (LCU) aimed at normal CPU processing, and (ii) The Throughput Compute Unit (TCU) aimed at parallel processing. The LCU supports the CPU's native instruction set (e.g x86-64 or ARM), as well as the HSA Intermediate Language (HSAIL) instruction set. HSAIL is translated into the native instruction set before it executes on the LCU, whereas the TCU executes HSAIL instructions only. The LCUs and TCUs can queue work onto each others' devices (e.g A GPGPU can request a CPU to perform some work and vice versa).

2.2.2 Accelerating Applications

There are two commonly used methods to access accelerators. The first uses Application Programming Interfaces (APIs) while the second uses compiler directives. In the former case, APIs such as the Compute Unified Device Architecture (CUDA) [33], OpenCL [34] and GPU Ocelot [35] provide a set of interface libraries that help hide the complexity of moving data between the machine (or *host*) and the accelerator (*device*); executing the accelerator code (*kernel*); and moving the results back to the host. In the latter case, the application developer will use "hints" in the application source code to suggest how the compilation of the code should be handled. These hints can, for example, direct the compiler to treat certain types of serial loops as parallel operations.

CUDA was developed for GPGPU computing on Nvidia hardware. It is not a portable application development interface and will not work with other vendor's GPGPUs, such as AMD and Intel. To execute instructions on an Nvidia GPGPU the application developer defines a lightweight *kernel*. The kernel code looks like a segment of a serial application (i.e. there is no direct reference to parallel execution) with the exception that the kernel will refer to a variable that indexes a thread identifier. A simple example program that adds two index-wise paired elements of two 1-dimensional arrays (vectors) using CUDA is illustrated in Listing 2.1.

Listing 2.1: Addition of two vectors in CUDA

```
// Kernel definition
__global__ void AddVectors(float* InputVector1, float* InputVector2,
                           float* OutputVector)
{
    int i = threadIdx.x;
    OutputVector[i] = InputVector1[i] + InputVector2[i];
}

int main()
{
    ...
    // Kernel invocation with N threads -- indicates size of vector.
    AddVectors<<<1, N>>>(Vector1, Vector2, Result);
    ...
}
```

CUDA's lack of portability to other accelerators is one of its biggest weaknesses.

Where application portability across a range of different accelerators is desirable, APIs such as OpenCL and OpenACC are a better choice.

OpenCL is an evolving set of standards that have been defined with the backing of a diverse range of hardware vendors, software development companies, research and development companies and academia. The goal is to support cross-platform parallel computing on a wide range of hardware including Nvidia, AMD and Intel GPUs, Intel's Xeon Phi, Intel and AMD x86-based CPUs, ARM-based devices and some FPGAs [36]. Furthermore, OpenCL defines a hierarchy of Vendor, Platform and Device that allows the application to use all OpenCL compatible devices simultaneously or to allow the application select particular hardware based on the vendor (e.g AMD), platform (e.g. CPU, GPU or both) and the device (e.g. the second of four AMD GPGPUs). Listing 2.2 illustrates the Vector addition kernel written in OpenCL. As with Listing 2.1, the OpenCL kernel code assuages the complexity of the defining parallel operations. Indeed, the Nvidia and OpenCL kernels are almost identical, a minor difference being how the data elements are indexed.

Listing 2.2: Addition of two vectors in OpenCL

```
__kernel void AddVectors(__global float4* InputVector1,
                         __global float4* InputVector2,
                         __global float4* Result,
                         unsigned int vector_size) {
  int ThreadID = get_global_id(0);
  if (ThreadID < vector_size)
    Result[ThreadID] = InputVector1[ThreadID]+InputVector2[ThreadID];
}

int main()
{
    ...
    // OpenCL Kernel invocation with N threads -- indicates size of vector.
    AddVectors(Vector1, Vector2, Result,N);
    ...
}
```

The OpenACC standard defines a set of compiler directives. These directives help specify where regions of application code written in C, C++ or Fortran (such as loops)

may be offloaded from a host CPU to the accelerator. OpenACC's design and philosophy are based on the success of OpenMP, which hides the complexity of writing parallel code for multi-core SMP machines. Like OpenCL, OpenACC is designed for portability across operating systems, host CPUs, and a wide range of accelerators. OpenACC is supported on Nvidia and AMD GPGPUs/APUs, as well as the Xeon Phi. Unlike CUDA or OpenCL, OpenACC allows programmers to develop hybrid CPU and accelerator applications without the need to explicitly initialize the accelerator, manage data or program transfers between the host and accelerator, or initiate accelerator start-up and shut-down.

Listing 2.3: Addition of two vectors in OpenACC

```
#define SIZE 100000
float Vector1[SIZE];
float Vector2[SIZE];
float Result[SIZE];
# an iterator variable
unsigned int i;
    ...
int main()
{
    ...
#pragma acc kernels copyin(Vector1,Vector2) copy(Result)
for(i=0;i<SIZE;i++)
{
  Result[i] = Vector1[i] + Vector2[i];
}
    ...
}
```

2.2.3 LRMS Support for Accelerators

In a shared cluster (see Section 2.1) the principal resource managed by all LRMSs is the CPU. In addition to the allocation status of the CPU, the choice of worker node on which to schedule a job may also be influenced, for example, by a set of other factors, such as the number of CPU cores the job requires, the number of CPU cores available, the system load of the worker node, the disk space or memory available, or access policies.

Scheduling jobs that target accelerator resources can be handled in several ways.

The schedulers may allow the cluster administrators to define their own resource types and associate a number of those resources with a given worker node, i.e as a consumable resource (p. 16). Alternatively, the scheduler may specifically support the accelerator. While the latter is preferable from a resource management point of view, newer types of resources are not always supported. Although there have been some attempts to categorise and compare LRMSs [37] and their capabilities [38] [39], there does not appear to have been a comprehensive study analysing whether these LRMSs support accelerators. Table 2.1 attempts to fill that void. The table indicates whether the three most popular accelerators, namely the Xeon Phi, Nvidia GPGPUs and AMD GPGPUs, can be specifically micro-managed by the LRMS, or whether there is a method to manage them as a type of consumable resource.

Table 2.1: Overview of LRMS for Accelerators

LRMS	Xeon Phi	Nvidia	AMD	Consumable Resource
TORQUE	Yes	Yes	No	Yes
SLURM	Yes	Yes	Yes	Yes
Sun Grid Engine	No	No	No	Yes
HTCondor	No	Yes	Yes	Yes
LSF	Yes	Yes	Yes	Yes

The problem with using a consumable resource is that the LRMS does not actually micro-manage the resources themselves. The LRMS will only manage the number of resources allocated from each node. Furthermore, the user also needs prior knowledge about the (arbitrary) name given to that consumable resource and how it is used in the job specification.

While micro-management may not be a problem on clusters that only have only one accelerator of each type on its worker nodes, it is a problem when there are several accelerators of the same type on the same worker node – there is no micro-management to assign individual accelerators to the job [40] [41] [14]. This may lead to inadvertent resource contention.

Some accelerators provide additional software-based techniques that can be used to manage and control user access to allocated resources. The current practice is to use an "environment" variable to specify the list of resources that are available to the user. The environment variables that correspond to particular accelerators are listed in Table 2.2. The relevant variable is interrogated at runtime when the user's

accelerator-based application executes any call to the accelerator's runtime library (e.g. CUDA, OpenCL). Only the devices assigned to the variable are visible and accessible to the user, all other accelerators are hidden. The format of the variable's value is a comma-separated list of integer values, and each integer uniquely corresponds to an accelerator. Table 2.3 lists whether popular LRMSs automatically enforce accelerator assignment through the use of environment variables.

Table 2.2: Environment variables supporting accelerator visibility

Accelerator	**Environment Variable**
Xeon Phi	OFFLOAD_DEVICES [42]
Nvidia	CUDA_VISIBLE_DEVICES [43]
AMD GPGPU	GPU_DEVICE_ORDINAL [44]

Table 2.3: LRMS Accelerator support using environment variables

	Xeon Phi	**Nvidia**	**AMD GPGPU**
TORQUE	Yes[1]	Yes[2]	No
SLURM	Yes	Yes	Yes
Grid Engine	No	No	No
HTCondor	No	Yes	Yes
LSF	Yes	Yes	?

Actual LRMS accelerator support is more complex than these tables show. For example, to enable full GPGPU resource management and monitoring in TORQUE, the TORQUE software must be recompiled and linked against the TESLA Software Development Kit [45]. Even then, this is only supported by particular high-end NVIDIA GPGPUs. Integration of cheaper mid-range GPGPUs under TORQUE is not fully supported. In addition, with older versions of TORQUE, the user was responsible for setting the $CUDA\_VISIBLE\_DEVICES$ variable themselves [46]. Furthermore, there is no official support for GPGPU resources under TORQUE when it is combined with the more advanced MAUI resource manager. This presents a significant challenge for Grids, as the combination of TORQUE and MAUI is the most popular LRMS deployed on the Grid (see Section 2.4.4). A solution to this problem is presented in Chapter 3.

[1] From Version 4.2.2 onwards

[2] from Version 4.2.10 – back-ported from 5.1.1

2.2.4 Accelerated Computing: Top 500 Case Study

The biannually published Top 500 High Performance Computing Systems [47] list ranks the fastest computing systems in production use worldwide based on a standard Linpack benchmark [48]. Although the list is concerned with the pinnacle of high performance computer systems, it is often seen as a key indicator of the type of systems that filter down to non-Top500 research and commercial resource centres within a five to ten year time span. For example, the first Top500 system to achieve 1.6 teraflops was the Intel ASCI Red [49] in 1997. This system consisted of almost 10,000 processors. However, by 2009 a single PS3 games console was capable of executing 218 gigaflops [50] – a single machine performed almost one eighth of the work that the world's fastest machine performed just over a decade previously.

The first contemporary accelerator-based system made it onto the Top 500 list in 2006. This system used the ClearSpeed CSX600 [51], a PCI-e card that supported 96 cores called *Processing Elements*. This was followed in 2008 by a small number of IBM Cell-based systems. The first set of Nvidia GPGPU-based systems made their appearance on the Top 500 list in November 2010, however it was only after June 2011 that accelerated systems started to have a significant impact, growing from nineteen accelerated systems in June 2011 to 88 in June 2015 (Figure 2.4). It should be noted that although Nvidia GPGPUs have been the predominant accelerator in use over this period, in 2015 there has been a drop in the number of systems using Nvidia (51) and a sharp rise in the number of systems using Intel's Xeon Phi (33), and three systems are deployed with both Nvidia and the Xeon Phi.

To determine if there are further trends in the growth of systems using accelerators, it is worth examining what patterns can be seen within the (cumulative) subgroups of the the Top 10, 20, 100, 200 and 500 systems respectively. The data is, again, derived from the Top 500 list over the period from June 2011 until June 2015 and shows the number of systems in each subgroup (Table 2.4) and the percentage of the systems that use accelerators in each subgroup (Table 2.5). An alternate graphical view of the data from Table 2.4 data is illustrated in Figure 2.5. Two trends are visible from the data, clearly showing that the number of accelerated systems in the Top 10 has remained relatively constant (The IBM BlueGene/Q system is also heavily represented in the June 2015 list, with four such systems in the Top 10), so the growth has been outside the Top 10; and in the Top 500 subgroup there has been an increase from 3.8% of

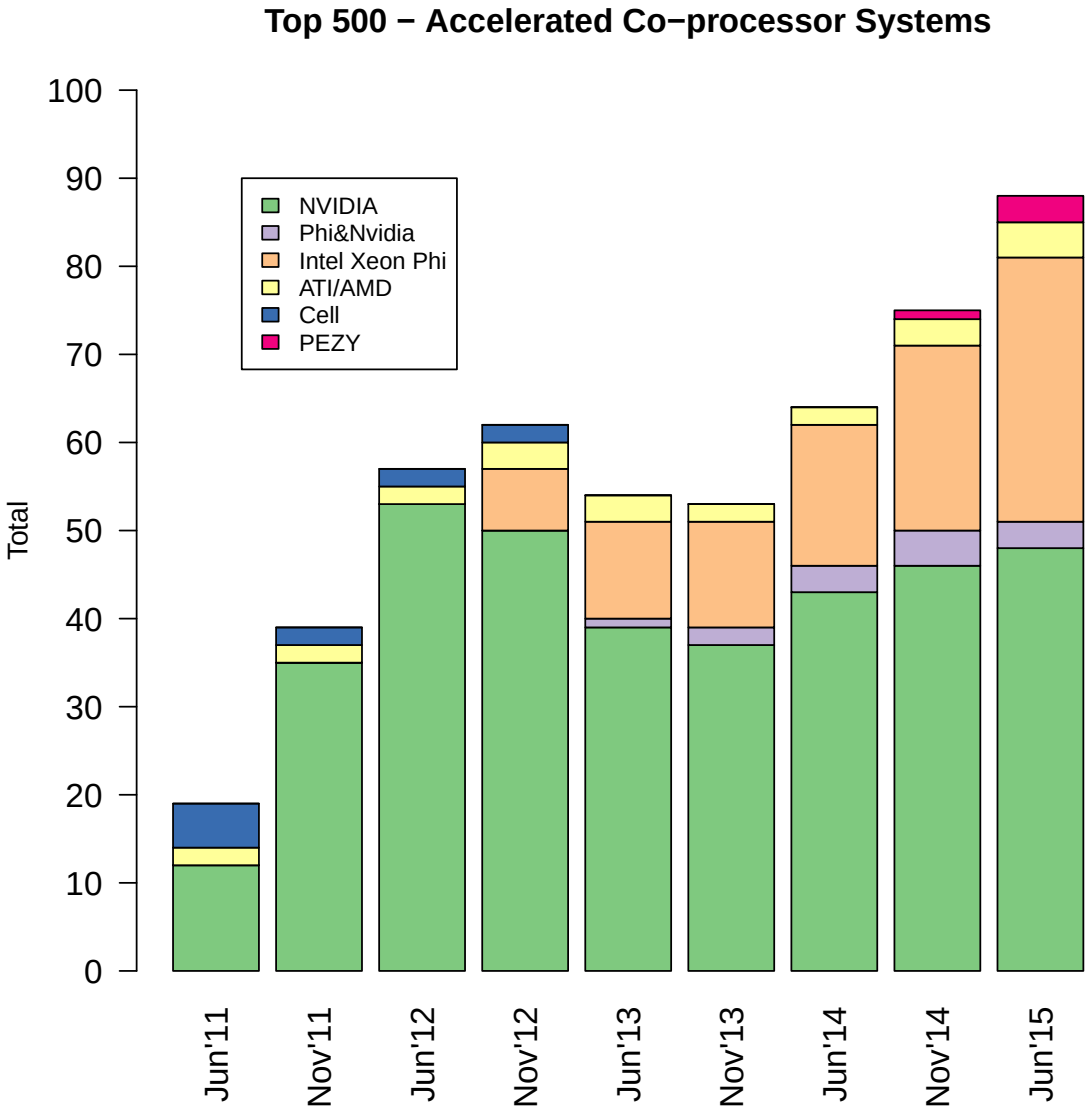


Figure 2.4: The Growth of Accelerated Systems by type in the Top500 (2011-2015)

	Jun11	Nov11	Jun12	Nov12	Jun13	Nov13	Jun14	Nov14	Jun15
Top 10	4	4	3	3	4	4	4	5	4
Top 20	5	6	5	5	5	7	8	9	8
Top 100	10	12	16	20	24	25	27	28	26
Top 200	13	31	32	30	33	39	43	45	49
Top 500	19	39	58	62	54	53	64	75	88

Table 2.4: The number of Top 500 supercomputers using Computational Accelerators within the Top 10, 20, 100, 200 and 500 subgroups.

	Jun11	Nov11	Jun12	Nov12	Jun13	Nov13	Jun14	Nov14	Jun15
Top 10	40%	40%	30%	30%	40%	40%	40%	50%	40%
Top 20	25%	30%	25%	25%	25%	35%	40%	45%	40%
Top 100	10%	12%	16%	20%	24%	25%	27%	28%	26%
Top 200	6.5%	15.5%	16%	15%	16.5%	19.5%	21.5%	22.5%	24.5%
Top 500	3.8%	7.8%	11.6%	12.4%	10.8%	10.6%	12.8%	15%	17.6%

Table 2.5: The percentage of Top 500 supercomputers using Computational Accelerators within Top 10, 20, 100, 200 and 500 sub-groups.

systems using Accelerators in June 2011 to 17.6% in June 2015.

If the sub-ranges of Top 100, 101-200, 201-300, 301-400, 401-500 systems are considered, then the histograms of Accelerator deployments (Figure 2.6) show that while Accelerators have been mostly deployed in the Top 100 and Top 200 systems, the most recent June 2015 data shows that the distribution is becoming more evenly spread across each sub-range. Although this may indicate growth of Accelerated systems outside the Top 500 (by inference), the lack of sales data for the Intel Xeon Phi and server-class Nvidia GPGPUs make it difficult to confirm this.

The increasing importance and popularity of using accelerators for massively-parallel processing in the computational sciences is clear [4] [5]. It is likely that the use of such accelerators, as exhibited in the Top 500 computing systems, will also be replicated to other computing environments such as Grid. However, as there are several diverse types of competing technologies, the challenge will be to accommodate this diversity.

2.2.5 Benchmarking Accelerators

The aim of benchmarking is to provide an independent assessment of how the performance of (parts of) a system compare to a predefined standard. For example, knowing the performance of a processor is crucial to determining the expected amount of time that certain (known) applications may take to execute. Benchmarking may be used as a simple way to extrapolate the time it will take to run a given application on

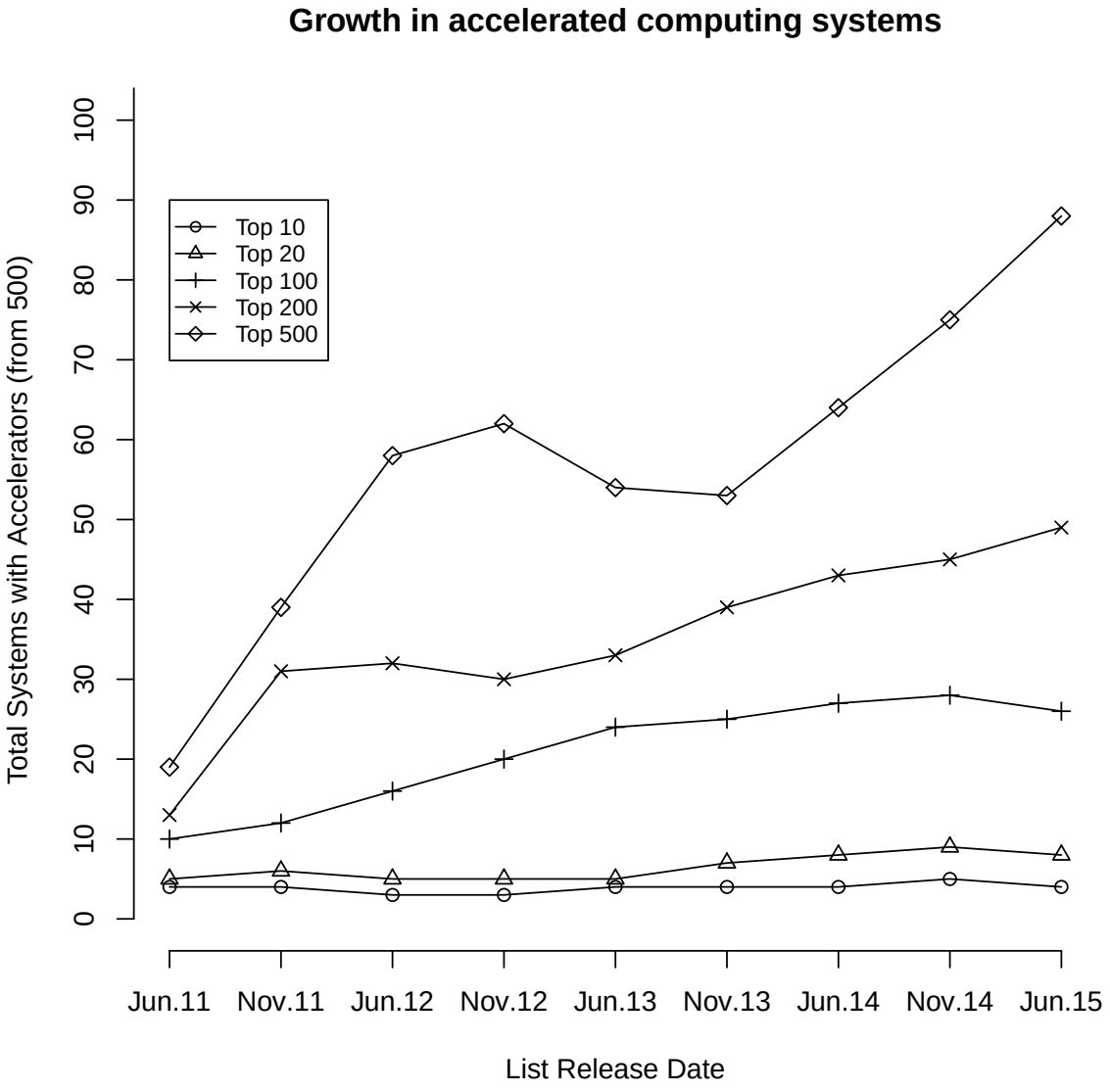
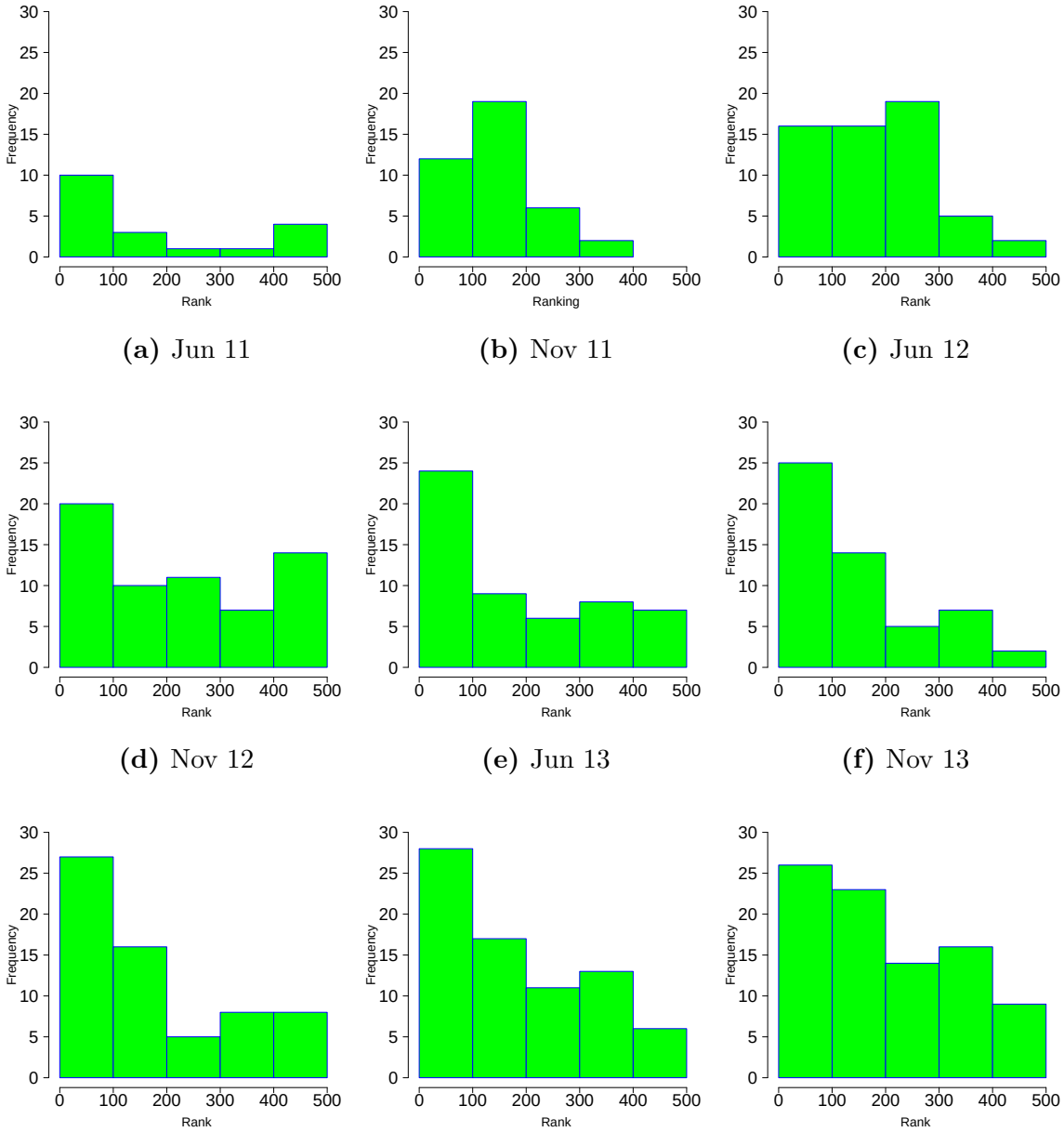


Figure 2.5: The Growth of Accelerated Systems sub-grouped into Top 10, 20, 100, 200 and 500 (2011-2015)



(a) Jun 11 **(b)** Nov 11 **(c)** Jun 12

(d) Nov 12 **(e)** Jun 13 **(f)** Nov 13

(g) Jun 14 **(h)** Nov 14 **(i)** Jun 15

Figure 2.6: Frequency distribution of Accelerators in Top 500 (June 2011 to June 2015)

different processors. Furthermore, by having a standardised way to distinguish how different processors perform relative to each other, benchmarking provides key information that allows users or resource management systems to choose the best possible resource available at a given time.

The Standard Performance Evaluation Corporation (SPEC) SPECint and SPECfp benchmarks are the de-facto standard suites for measuring the integer and floating-point performance of the CPU respectively. As processor performance and memory capacity has rapidly grown, the SPEC benchmark has been regularly updated to reflect those changes. The latest version of both SPECint and SPECfp is generally referred to as SPECint 2006. This benchmark suite is widely used by the grid computing community to (i) allow users to locate resources with an expected performance level; and (ii) to allow the resource providers to quantify the computational resources they have provided over a given time period (accounting).

SPEC have only recently (April 2014) released their first benchmark suite that is aimed at gauging the compute capabilities of accelerators – the SPEC ACCEL benchmark suite [52]. The suite consists of two distinct sets of benchmarks: the first is based on OpenCL (version 1.1), and the second on OpenACC (version 1.0). The use of two APIs ensures that the benchmarks can handle a wider range of accelerators and application requirements. The OpenCL suite re-uses a number of existing application benchmarks (notably Parboil [53] and Rodinia [54]) from a variety of scientific domains such as linear algebra, image processing, bioinformatics and physics. These help measure accelerator performance under different application conditions and requirements (speed, memory, data-movement patterns). Furthermore, they also examine aspects of the performance of the system, such as the bandwidth between the host and the accelerator, as well as software related aspects (compilers and drivers).

Due to its recent release, SPEC ACCEL is cited in a relatively small number of research publications. Indeed, at the time of writing, the only publication that was cited in IEEE Xplore [55] or Google Scholar [56] was a review of the benchmark itself [52], and this has no known citations to date. Instead, the most widely cited benchmark for accelerators is the Scalable HeterOgeneous Computing (SHOC) [57] benchmark. This implements both CUDA and OpenCL versions of its benchmark applications, allowing it to execute on a wide range of accelerators, and allowing it to compare the performance of the two APIs executing on the same Nvidia based GPGPUs. Similar to SPEC ACCEL, SHOC provides a suite of benchmarks to gauge hardware performance

and examines the data bandwidth between the host and the accelerator.

One factor that may be limiting the wider use of SPEC ACCEL is its pricing. Even with a substantial discount for academic institutions, the educational cost of the benchmark is $800 [58], whereas SHOC is free and open source.

2.3 Virtualisation

One of the objectives of this thesis is to extend the range of discoverable accelerated computing services available on grid infrastructures. In Chapter 4 it will be shown how a service offering access to virtualised accelerated computing resources – virtual GPGPUs – can be integrated into the grid. This will be based (and expands) upon the methods developed in Chapter 3 for integrating physical resources.

In this section, the topic of machine virtualisation is presented. This is intended to provide a background into the techniques that are available to create virtual machines that have access to computational accelerators. Mechanisms for virtualising the GPGPU itself, both with and without machine virtualisation, are discussed in Section 2.3.2.

2.3.1 Machine Virtualisation

Machine Virtualisation is the simulation of the software and/or hardware components upon which other software runs [59]. This simulated environment is known as a *virtual machine* (VM). The VM is also known as the *Guest Operating System* or *guest OS*. Machine Virtualisation is fundamental to the Cloud Computing paradigm, which seeks to make computing services a consumable product that is on-demand, flexible and scalable according to user needs.

Several models of machine virtualisation have emerged, each with their own strengths and weaknesses. These models are classified as follows:

Full Virtualisation uses the physical machine to execute a special service, known as the *Virtual Machine Monitor* or *Hypervisor*. This service acts as a supervisor process that co-ordinates the interaction between the VM and the physical hardware. Under full virtualisation the VM is constructed and stored either directly as an encapsulated image file on a disk, or it may be constructed to use a physical

disk directly. The VMs execute on the same CPU architecture as the hypervisor, and this requires that the VM must be compatible with the physical CPU. Furthermore the physical CPU may need to support additional instructions that are not part of the core specification (e.g. Intel's X86 and AMD both require additional CPU instruction support). A major advantage of full virtualisation is that all running VMs execute their own individual OS and Kernel – thereby allowing each machine to execute in isolation from one another. In this way, a single physical machine can host the simultaneous execution of many VMs with differing operating systems (Linux, Microsoft Windows, FreeBSD etc.). All of the guests support the same hardware, which allows the guest to execute many instructions directly on the hardware – thereby providing improved performance.

Further advanced VM techniques such as *PCI-Passthrough* allow a VM to take control of a physical piece of hardware (e.g a network card or GPU). However, while this is desirable, it requires additional support from (i) the hypervisor and virtual machine software, (ii) extended CPU instruction set support, (iii) the motherboard of the physical machine, and (iv) the hardware itself.

This is one of the most popular forms of machine virtualisation.

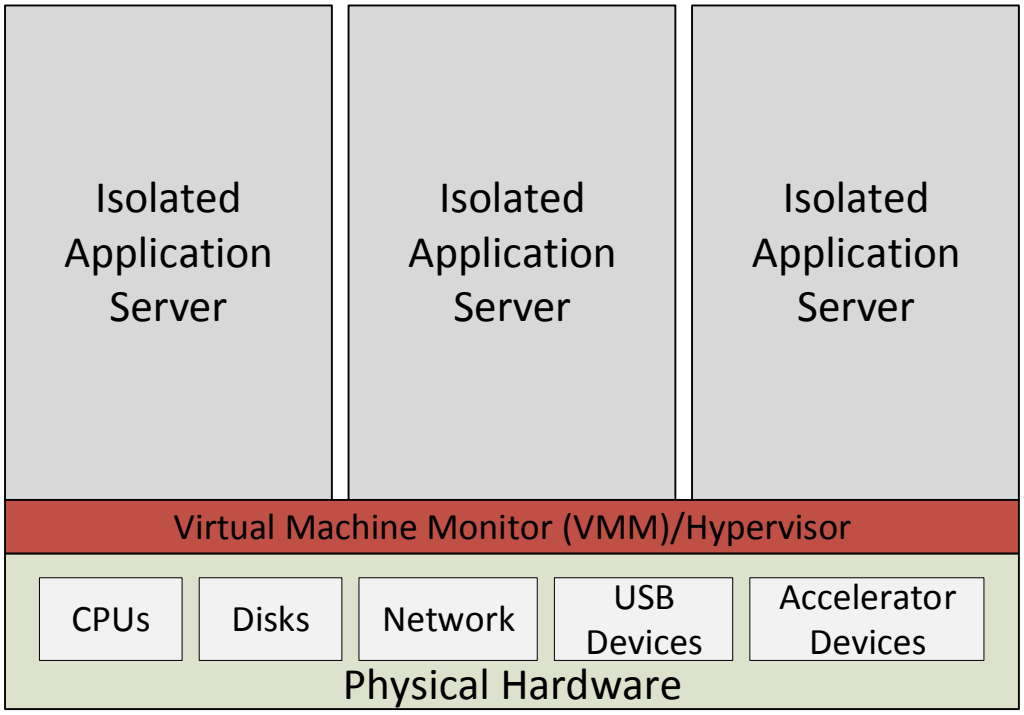


Figure 2.7: A representation of full-machine virtualisation

Para-Virtualisation is quite similar to full virtualisation in that it uses a hypervisor

to manage the interaction between VMs and the physical hardware, but there are two significant differences: (i) the hypervisor can support VMs without the expanded CPU instruction set, so it can run on a greater range of hardware; but (ii) The VMs must instead execute with a modified kernel, so some OS distributions may not be easily supported.

OS-level virtualisation doesn't strictly create a VM, but rather creates an isolated environment that contains and restricts a set of related processes. Such restricted environments are often called *Containers*. Containers share the operating system kernel with the machine that hosts it (which may itself be a VM), and as such, they share common drivers and devices with the underlying OS. Advanced systems for managing Containers can create environments that appear to be completely separate machines. Each instance may have its own file system, IP address, and server configuration. For this reason, some publications class them as VMs.

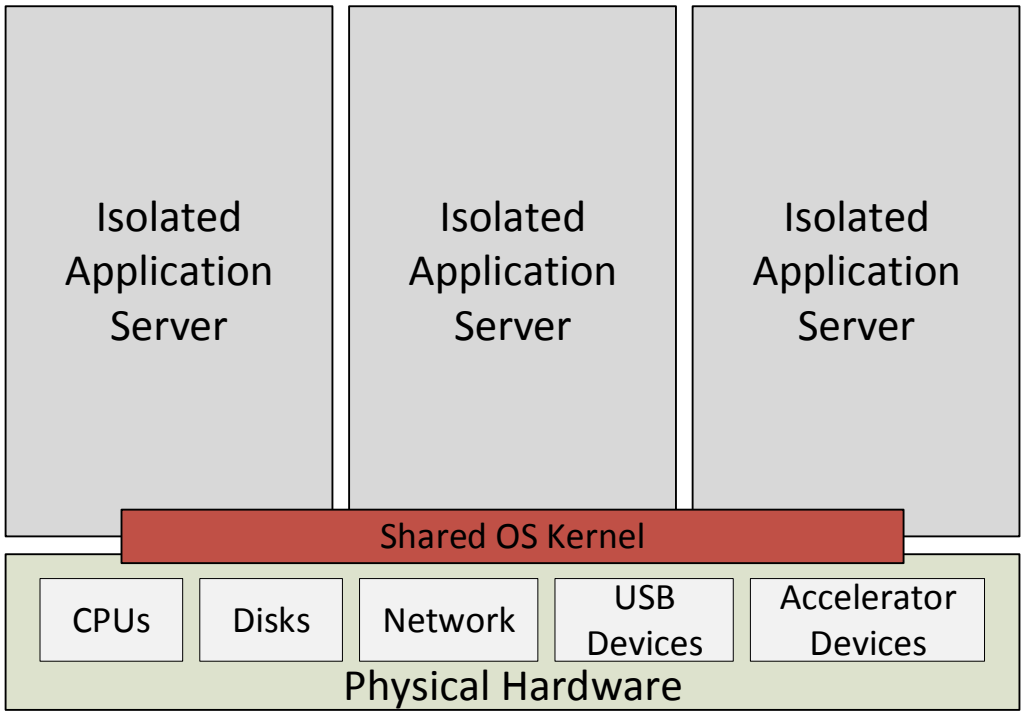


Figure 2.8: A representation of OS-level (Container) Virtualisation

Emulation is a form of machine virtualisation simulates the virtual environment (both software and hardware) completely in software. One of the key strengths of emulation is that it can simulate alternative CPU architectures or other hardware devices (including hypothetical and defunct hardware) than those available on the

physical machine – this allows software developers to create and test applications for alternative hardware (e.g mobile phone application development). However, such emulation tends to be slow in comparison to the real physical hardware.

OS, Full, and Para-virtualisation architectures are interesting to this thesis because they have the ability to pass some individual hardware devices (like accelerators) into the environment of a VM or Container. Full and Para-virtualisation can use PCI-Passthrough, whereas OS-level virtualisation simply passes the OS hardware device into the Container.

2.3.2 GPGPU Virtualisation

Resource- and data-centre machine virtualisation focusses on the common scenario of a typical (virtual) machine with CPU, disk storage and network access. However, there is also great interest in facilitating other types of hardware, especially in the domain of Desktop Virtualisation, where graphics cards, keyboards, etc. (that makes user-interaction possible) are virtualised. In this section the topic of GPGPU virtualisation is considered. This is an active research area and several different approaches have been studied. Virtual GPGPUs have the potential to change the resource requirements of GPGPU applications. For example, mobile devices can use them to offload computationally intensive applications, such as Computer Aided Design, to more powerful remote GPGPUs [60]. Virtual GPGPUs have also been used to execute scientific applications in a cloud-like environment [61].

As noted, several methodologies have emerged that enable some level of GPGPU virtualisation, and these can be classified into four categories:

API-Interception is a software method for virtualising GPGPU resources. GPGPU hardware is normally accessed through calls to an API such as CUDA or OpenCL. These calls may be *intercepted* (or *hooked*) before being directed to a physical GPGPU. This technique is used to provide transparent access to remotely installed GPGPUs as if they were local. Remote virtual GPGPUs use a *frontend/backend* model in which the *frontend* intercepts all API calls (and their data) and transfers them over the network to a selected *backend* for execution. Several API-Interception implementations have been developed for both CUDA (rCUDA [62], GridCUDA [63],vCUDA [64], gVirtus [65]) and OpenCL (VCL [66],

dOpenCL [67], SnuCL [68],vOpenCL [69]) runtime libraries. However, some have not been actively developed in recent years and do not implement recent changes to their respective APIs.

The drawback of API-Interception is that access to the backend incurs a performance penalty: two API calls (frontend and backend) for each regular API call; network data transfers and network latency. Performance studies of rCUDA [62] and VCL [66] examine some commonly used GPGPU benchmark suites (e.g. SHOC [57]) and e-Science applications (e.g LAMMPS [70]) to identify conditions where using API-interception is beneficial.

Kernel Device Passthrough allows a GPGPU device (e.g /dev/nvidia0) to be passed into a VM, and has the advantage of working with all GPGPUs. This method is easy to exploit using *containers*. Impact on performance is negligible – 0% for Containers [71] and 3% for gVirtus [72].

Kernel Device Passthrough by itself cannot provide a running application with access to more GPGPUs than are available on the local host. However, it will be shown in Chapter 4 of this thesis how to exploit this form of GPGPU virtualisation in conjunction with API-Interception to help provision a pool of isolated, remote virtual GPGPUs for use within HPC and Grid environments.

GPGPU isolation is a concern as the hypervisor and its containers (or VMs) share the same device. To ensure GPGPU isolation between containers, care must be taken to ensure no two containers are assigned the same device. A further concern with certain GPGPUs (e.g. Nvidia devices) is that, on a multi-GPGPU host, if a single GPGPU is passed through to a container, then all GPGPUs become available to that container. This issue will be addressed in Chapter 4.

PCI-Passthrough allows physical hosts to cede control of PCI-devices to a VM. This requires hardware support on the CPU, GPGPU, motherboard, and VM hypervisor. Although support for PCI-Passthrough has improved recently, with many hypervisors now supporting it, there are still relatively few GPGPU devices with support for the facility. A recent study [71] concluded that PCI Passthrough has negligible performance impact.

PCI-Switching is a low-level technology that allows a physical machine equipped with additional specialist hardware to be assigned external PCI devices attached

to the switch. Although, this technology allows flexible hardware reconfigurations, equipment cost is a significant disadvantage. It can be used in conjunction with PCI-Passthrough (dependent on suitable hardware) for additional flexible VM configurations.

Machine Virtualisation and GPGPU virtualisation will be used for two primary purposes in this thesis: the first is to aid GPGPU resource allocation and access control; while the second will be to construct user-configurable environments with access to many (virtual) GPGPUs. The latter is made possible by combining (layering) multiple virtualisation technologies. For example, a GPGPU encapsulated and assigned to a VM (using Kernel Device Passthrough, PCI-Passthrough or PCI-Switching) can be accessed from a remote worker node using API-Interception. The use of multi-layered virtualisation (applied to GPGPU resources) will be studied in Chapter 4.

2.3.3 Cloud Computing

Cloud Computing is a computing paradigm that supports convenient and on-demand access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services). These computing resources can be rapidly provisioned and released with minimal management effort or service provider interaction [73]. The paradigm attempts to optimise utilisation of physical machine resources by allowing CPU, disk and other resources to be assigned to one or more Virtual Machines (VMs). The VMs have their own independent machine environment which improves resource and job isolation, and they can be highly customised to execute specific user applications or services. As such, the Cloud Computing model is primarily for loosely-coupled, on-demand (elastic), or highly-scalable parallel computing computing tasks (e.g Map/Reduce [74]) or services.

Clouds may be abstracted as cluster computing with elasticity, although the concrete details may be somewhat different. In contrast, large-scale Clouds and HPC systems are quite different. They cater to different types of applications (i.e. loosely-coupled application in custom environments vs tightly-coupled applications on monolithic systems). Nevertheless there are some similarities at the infrastructure and systems levels. Both models are aimed at handling very intensive use of computational resources. They are typically set in specialised facilities that provide adequate redundancy for critical services such as power, cooling and high-bandwidth structured

networking, and the machines are maintained by a specialised team of operators. Furthermore, it is typical for both these computing service providers to consolidate as much hardware capacity and capability into the smallest feasible number of physical machines, including densely packing accelerators into the machines. Given the similarities, it is no surprise that some Cloud Computing providers, such as Amazon, cater for the provisioning of accelerator hardware, including GPGPUs [75]. Indeed, some economic models have shown that the cost of running some HPC applications in a Cloud environment may be significantly cheaper than running the same applications in a dedicated HPC environment [76]. There is an argument for the provisioning of HPC-on-Cloud, and this has been championed in several case-studies [77] [78]. However, the counter-argument is that certain machine virtualisation models, such as full virtualisation (p. 36), can decrease application performance [79].

Some of the key differences between Cloud and HPC systems are evident in how those resources are used, and the types of applications that execute on them. HPC systems, unlike Cloud, focus on delivering the optimal computational power and capability possible, thereby allowing user jobs to execute as quickly as possible. These systems may be shared among hundreds or thousands of users from different scientific backgrounds and with different application needs. Such sharing requires management tools to optimise utilisation without overcommiting resources, and this is typically the responsibility of an LRMS.

Related to the HPC-on-Cloud model is Cloud-on-HPC, i.e. bringing Cloud-like services and custom application environments into existing HPC environments. However, as HPC systems tend to be highly customised monolithic systems, it is not always practical or desirable to integrate cloud solutions (e.g. OpenStack [80] and OpenNebula [81]) into pre-existing HPC environments.

In this thesis, an alternative lightweight solution that uses OS-level virtualisation (p. 38) and avoids the performance degradation of full virtualisation and the difficulty of integrating complex Cloud middleware into existing clusters, HPC systems and Grids shall be developed. The objective will be to virtualise accelerator resources, with a view to improving their utilisation in cluster and grid environments (see Chapter 4).

2.4 Grid Computing

Cluster-based computing can provide sufficient computing power for many types of computational-based problems. However, the main issues with Clusters include: (a) the user must have a local account; (b) there may be significant physical or financial constraints that limit the size of the cluster – this will limit the scope of the problem that can be investigated; (c) there is a financial cost to under utilisation – for example, each node consumes power even when idle, reducing value-for-money and return on investment; and finally, (d) over-utilisation means local users may not have access to compute resources when they need it. Furthermore, for very large-scale computational science problems, the computational capability of a single Cluster may not be sufficient. Indeed, many large scientific experiments (e.g. the Large Hardron Collider [82] at CERN, or the Square Kilometre Array [83]) can now only be realistically investigated by pooling the skills and resources of many different universities or research institutes (partners or administrative domains) in the form of distributed research collaborations. As partners contribute their own set of computational resources and skills, there are logistical and administrative issues in ensuring that each individual has (sufficient) access to the pooled resources and that these shared resources are efficiently managed. Grid Computing developed out of the need to solve these issues by providing the technical and administrative means to (a) pool resources across administrative domains; (b) provide a way for any user to be uniquely and verifiably identified; (c) allow collections of users to form virtual groups, where each group has a specific common purpose and goal; and (d) identify the resource usage at each resource centre, and allow the user to send jobs to under-utilised resources where possible.

2.4.1 Introduction to Grid Computing

The original concept for "Grid Computing" [1] was derived from the electrical grid utilities system consisting of customers (users) and electricity suppliers (resource providers). Both users and resource providers are inter-connected through a managed infrastructure (municipal, national or trans-national) that allows the customer to connect compatible pieces of electrical equipment into a wall socket and power them. The required power is transparently provided on-demand, and the user doesn't need to worry about how the power was generated. Although this idea was originally envisioned by the

computer science pioneer John McCarthy in the 1960's as a hypothetical way in which future computer-based services could be delivered [84], it wasn't until the late-1990's that the Grid, as proposed by Ian Foster and others, was conceived and realised through the Globus middleware [85]. Despite it being over a decade and a half since the definition of Grid was formalised [86], the terminology "Grid" and "Grid Computing" has a wide and varied interpretation [87]. With this in mind, there is a need to further clarify the use of these terms within the context of this thesis, and to set out the the scope and conditions that the work presented applies to.

This section introduces key Grid Computing concepts – specifically Grids such as the EGI and Open Science Grid (OSG) that primarily provide computing and storage resources to e-Science researchers. A high-level overview of a grid infrastructure and the life cycle of a user's job, from job submission through to execution using a Grid resource, will be presented. Grids are, by nature, large-scale, distributed infrastructures and accessing Grid resources relies on maintaining a uniform, structured, global view of the Grid. An overview of the systems that manage important infrastructure and resource information will also be presented. It also should be noted that the abstract idea behind Grid Computing overlaps with Cloud-based *Utility Computing*, but key significant differences, such as grid resources not being centrally controlled and Grids using open standards distinguish them [88].

Key Concepts

A Grid is a distributed collection of computational and storage resources where (i) resources are coordinated, but the resources are controlled and managed both solely and independently by its owner or *resource-provider*/*resource-centre* (for example a university, research centre, company or private individual). Each resource-provider has some level of control over how the exposed resources are accessed and used; (ii) the Grid uses open standards and general purpose protocols and interfaces; and, (iii) it delivers non-trivial qualities of service. This definition is sufficiently general to include both large-scale Grids, such as EGI or OSG, and also "compute-cycle" volunteer donation systems such as Berkeley Open Infrastructure for Network Computing (BOINC) [89].

The work described in this thesis is specifically concerned with large-scale Grids, such as the EGI or OSG, that are composed of many *resource-centres*. These Grids pool the resources of the participating resource-centres, and cater for thousands of dis-

tributed users from a variety of scientific disciplines. Furthermore, EGI and OSG are based on a number of common standards, such as the information model (used to describe their resource and services) and the security model (that is essential to building trust between all the participants – users and resource-centres alike). Common standards enable these Grids to interoperate with one another, and also allows individual resource-providers to participate in multiple Grid infrastructures. For example, some OSG resources can be accessed through the EGI, and the Grid-Ireland infrastructure was composed of eighteen resource providers, some of which also participated in the EGI.

The Grid Infrastructure

Each resource-centre in the Grid – a *Site* (Figure 2.9) – provides one or more *Compute Elements* (CEs) and one or more *Storage Elements* (SEs). The CE is the grid interface to access computational resources – namely clusters of worker nodes managed through an LRMS. The SE is a grid interface to a pooled collection of storage systems. The state of these Site services are regularly monitored and published through a user queryable Grid Information System. This information system is used to publish the capability and capacity of each resource-centre, in the form of a description of the resources that it provides, the current utilisation and availability of those resources and a description of the mechanisms for accessing them. The software that helps implement grid services is known as *grid-middleware* (or here, just *middleware*). There are several different middleware implementations available, including ARC [90], gLite [15], Globus and UNICORE [91] – each with their own set of service interfaces, grid tools and commands.

The collection of resources in a Grid form its foundation, with each Site providing computational and storage services. However, the primary difference between a Grid and a collection of Clusters is that the Grid resources are bound together by a set of higher-level services that facilitate discovery, sharing and utilisation of those distributed resources. Furthermore, the advantage of this type of system is that these higher-level services allow computational workloads to be distributed across the Grid to execute on under-utilised resources.

The set of services used in a basic grid infrastructure (Figure 2.10) may include: (i) the User Interface – a node from which a user submits jobs to the Grid; (ii) the

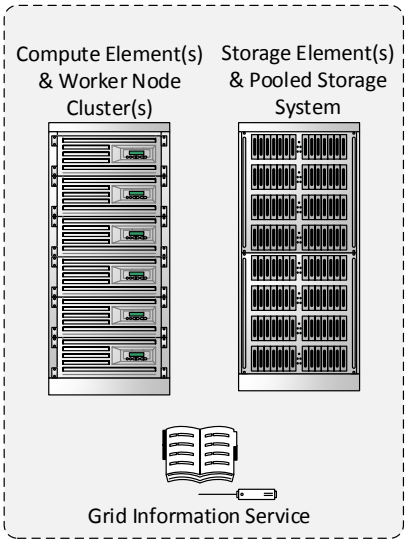


Figure 2.9: A simplified representation of a Grid resource-provider (*Site*), consisting of computational, storage and grid information resources

Grid Information System – a set of services for publishing the state of the Grid; (iii) a Workload Management System – a brokering service for matching grid-jobs, based on their requirements, to suitable resources and orchestrating those jobs; and (iv) Virtual Organisation management – a service that helps manage collections of users, and facilitates role-based access control to grid resources. Moreover, some grids such as EGI deploy supplemental grid-wide services that support distributed data catalogues. Data can be dispatched acrosss the grid with data redundancy support (fault tolerance) and all copies can be located using a file name alias.

To make use of the computational resources provided by a Grid, users submit *jobs* that are described using a formal *job description language*. Some of the languages are tightly-coupled with a corresponding middleware, for example Globus and RSL [92]. However, some middleware implementations may support one or more different job description formats (Table 2.6). In particular, both JDL [93] and JSDL [94] are supported across a range of middleware implementations.

A job description may consist of an executable program, a specification of the software environment in which the program must run, input parameters and data required by the program, and the name of output files produced by executing the job. Listings 2.4, 2.5 and 2.6 illustrate an example of a simple "Hello World" type

[1]Extended with ARC specific elements
[2]IGE BesGRAM Extension [95]

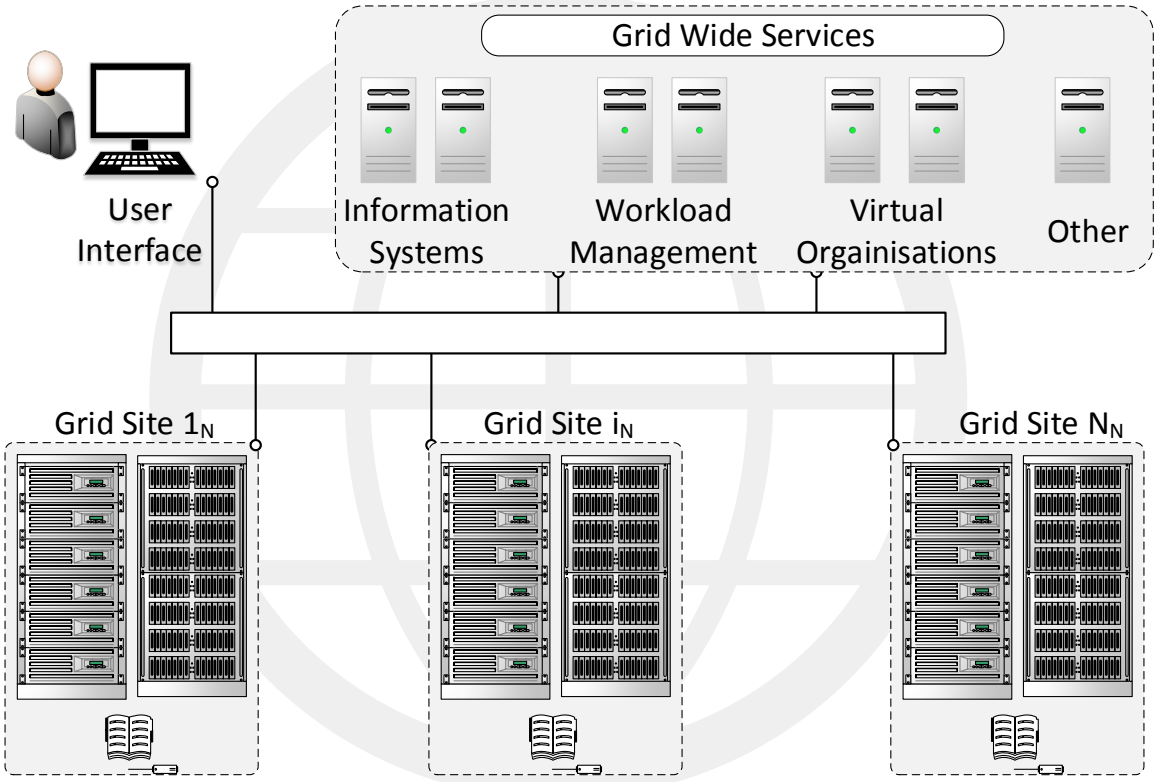


Figure 2.10: A Simple Grid Infrastructure

application in the various job description formats. It should be noted that, although it is possible to use the JSDL format in conjunction with most middleware, in practice the format is more difficult to use without additional visual tools to generate said JSDL.

Listing 2.4: Example JDL Hello World specification

```
[
Executable = "/bin/echo";
Arguments = "hello world";
]
```

Name	Acronym	Middleware
Job Description Language	JDL	ARC gLite
Job Submission Description Language	JSDL	ARC[1] Globus[2] gLite UNICORE
Resource Specification Language	RSL	Globus
Extended Resource Specification Language	xRSL [96]	ARC

Table 2.6: Middleware support for Job Description Language Formats

Listing 2.5: JSDL Hello World Application

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
 xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix">
 <jsdl:JobDescription>
  <jsdl:Application>
    <JobName>Test Job</JobName>
    <Description>Simple Hello World Job</Description>
    <JobProject>Test project</JobProject>
    <jsdl-posix:POSIXApplication >
       <jsdl-posix:Executable>/bin/echo</jsdl-posix:Executable>
       <jsdl-posix:Argument>hello world!</jsdl-posix:Argument>
     </jsdl-posix:POSIXApplication>
   </jsdl:Application>
  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

Listing 2.6: Example xRSL and RSL Hello World Application

```
(&(executable  = /bin/echo)
  (arguments   = "hello␣world")
)
```

Grid users can request that their jobs be executed on a specific resource, based on *a priori* knowledge of the capabilities of a resource-centre. This job submission method (or strategy) is of very little benefit to the normal grid user, because Grids are composed of a very large number of distributed resources, whereas a priori job

submission assumes that the user user is only interested in a specific resource, and not interested in choosing from amongst the other potentially better resources (where the job may finish quicker because there are more free CPU cores available or the CPU cores are faster, or the system has more memory).

Grids such as EGI, OSG and LCG provide high-level services that allow users to submit a job and let the Grid "decide" where that job should execute. Indeed, this job submission philosophy is in line with the Grid/Utility ideal that the end-user need not be (overly) concerned about how the resources are provided. In this scenario, instead of submitting a job directly to a resource-centre, the job is submitted to a Grid *Workload Management System* (WMS). The WMS acts as a broker, using the information published about each resource-centre through the Grid Information System, together with the description of the job, to find all resources that match the job's requirements. Furthermore, the broker can select one of the matched resources (e.g. using pre-defined policies or heuristics) and orchestrate the execution of the job on the chosen resource. Several implementations of WMS exist including the gLite WMS [97], ReSS [98], DIRAC [99], Panda [100] [101], ARC [90] and the glidein-WMS [102].

Finally, Users are grouped into *Virtual Organisations* (VO). These are usually based on their research area and/or projects that the user is involved with. These groupings are used to control access to – and account for the usage of – grid resources.

A Grid Job Life Cycle

The usual starting point for a user when submitting a job to the grid is the User Interface (UI). The UI is a service node that contains the necessary command tools to interact with other grid services. It is configured to interact with one or more Workload Management Systems (WMS). As noted in the previous section, there are several WMS implementations. Figure 2.11 shows an example that illustrates the life cycle of a single grid-job, from the submission of the job in JDL format on the UI, through its orchestration by a gLite WMS, to its eventual execution at a resource-centre. A high-level outline of the flow of a grid job through the WMS includes the following stages:

1. When the job is submitted to the WMS, a copy of the JDL file and any input files specified in it are copied to the Workload Management System (WMS).

2. The WMS will determine all locations where the job can possibly run. This is the

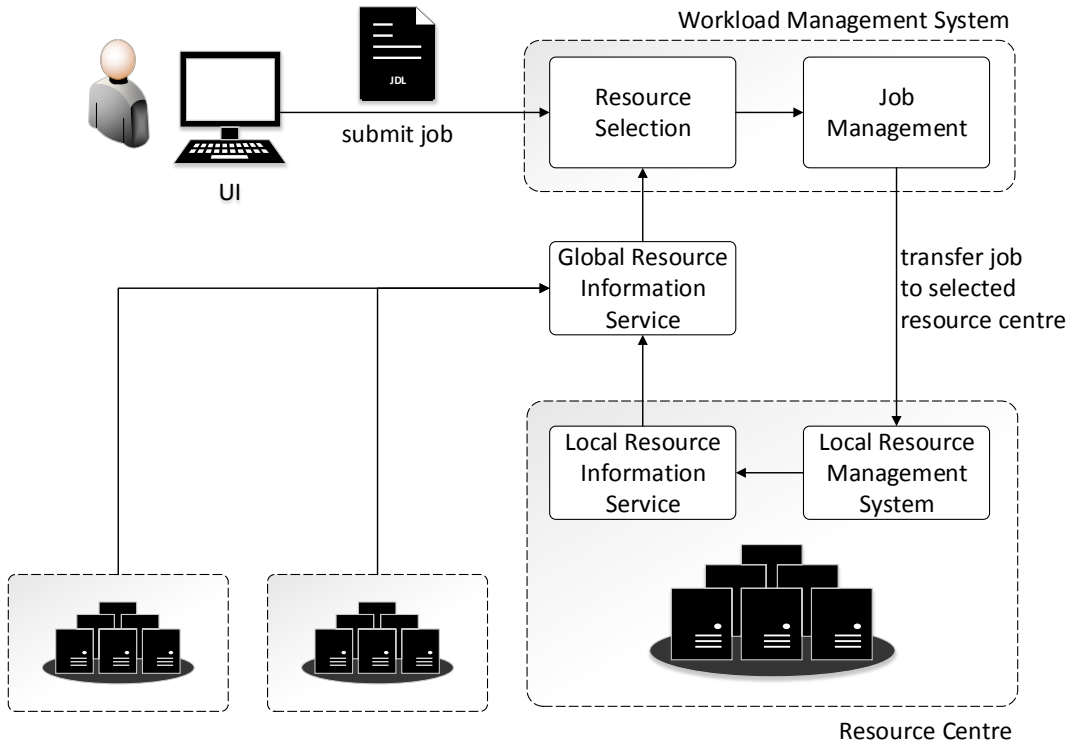


Figure 2.11: The Job Submission Chain

"match-making" process. The potential locations are then *ranked* in preference. The user can also influence the rank ordering by specifying a *RANK* expression in the JDL.

3. Once a target Compute Element (CE) at a resource-centre is chosen, the WMS will engage with the CE.

4. The CE builds a *JobWrapper*. This executable is built taking the *Local Resource Management System* (LRMS) used on the CE into account. Some of the roles of the JobWrapper are to build a job submission script for the LRMS, transfer all input files from the WMS, and then submit the job to the LRMS.

5. The LRMS is responsible for scheduling the execution of the user's job on one or more *CPU Cores* on a *worker node* (WN).

6. After execution, the output files specified by the JDL description are transferred back to the WMS and can be retrieved by the user through the UI.

The match-making process in step 2 above requires knowledge of the overall state of the distributed set of resources. This state information is managed by a "Grid

Information System".

GLUE Schemas and the Grid Information System

The Grid Information System is a critical component required for discovering services, determining the status of resources and using this information in the selection of resources for submitted jobs. This system is composed of an information model (a *schema*) for describing entities (such as computational resources and available software) and the relationship between those entities, as well as a "presentation layer" that publishes this information as a frequently-updated queryable service (a *realisation*).

The relationship between entities often takes the form of a hierarchical structure. For instance, a Grid is composed of a set of Sites (resource-providers); each Site may provide a set of one or more services (e.g. computational, storage, security, and grid job orchestration). There may be several instances of these grid services; for example, a Site may have several LRMS's, each managing their own clusters of homogeneous worker nodes. The worker nodes are further classified as belonging to one or more subcollections (queues), and these queues will have their own usage polices. Furthermore, the LRMS may implement policies to ensure that certain Virtual Organisations (p.49) have guaranteed access to these resources through fair share allocations (p.16).

The *Grid Laboratory Uniform Environment* schema (*GLUE schema*) was developed as an Open Grid Forum (OGF) "reference standard" for publishing the state of multidisciplinary Grids. There are currently two major (incompatible) versions of the GLUE schema in common use – GLUE 1.3 [103] and GLUE 2.0 [104]. It is important to note that these specifications define conceptual models, where the models show how entities (resources, services, security policies, etc.) in a Grid relate to one another, and which properties each resource should (mandatory) or may (non-mandatory) possess. Furthermore, the conceptual model is independent of the concrete data-format used by specific technologies. Current concrete data-formats include the LDAP, XML, SQL, and JSON schemas.

The first GLUE specification (GLUE 1.0) grew out of a need for many early grid infrastructures and middleware projects, such as DataTag, The European Data Grid (EDG), iVDGL [105], LCG, and Globus, to converge on a consistent description of globally distributed grid resources and services [106]. The Grids already conformed to a common security model – the Globus Security Infrastructure (GSI) – but some of

the Grids published differing information to describe the available resources and services. Convergence on a common resource description specification would greatly improve grid-interoperability, something that was required in order to solve many grand-challenge problems, such as confirming the existence of the Higgs boson [2] [3].

The initial specification was proposed in September 2002. However, further updates to the specification followed in April 2003 (GLUE 1.1), February 2005 (GLUE 1.2), and October 2006 (GLUE 1.3) in order to solve numerous problems with the specification itself or to enhance the specification. Each of these incremental changes were required to be fully backward compatible. This restriction was deemed to be a major constraint, as it limited the evolution of both the schema and the grid-middleware that used it [106].

It should be noted that each new revision of the GLUE schema requires changes to each of the grid-middlewares. As these middlewares are intended to provide robust "production use" of the grid infrastructures, there must be an assurance that disruptions to services are kept to a minimum. Consequently, changes to both the schemas and middlewares need to be rigorously tested. This testing and deployment process is both costly in effort and time-consuming. Furthermore, changes may also affect grid users if they need to adapt existing applications. Despite the ratification of GLUE 2.0 in 2009, it has yet to fully replace GLUE 1.3 as of late-2015. A major motivation behind the evaluation of the conceptual strategies described in Section 3.2 is to identify the provisions that exist in GLUE schemas that would facilitate a more flexible and dynamic approach to the integration of new resources, without requiring changes to the schemas.

Grids such as EGI are currently transitioning from GLUE 1.X to the GLUE 2.0 standard by running services that can utilise both standards. GLUE 2.0 offers many improvements over GLUE 1.3, including a richer description of grid entities and their state and the ability to easily publish extra details about grid-entities. Moreover, GLUE 2.0 was developed to improve inter-grid interoperability [107].

Information in a Grid Information System originates from *Information Providers*. The services that are used to access and manage grid resources (e.g. an LRMS managing a set of worker nodes) should provide an interface that allows the Information Provider for that resource to determine the properties and current state of the resource, transform this information into an appropriate GLUE entity and publish the information (Figure 2.12).

Propagating GLUE entities so that they are visible "globally" in a Grid Informa-

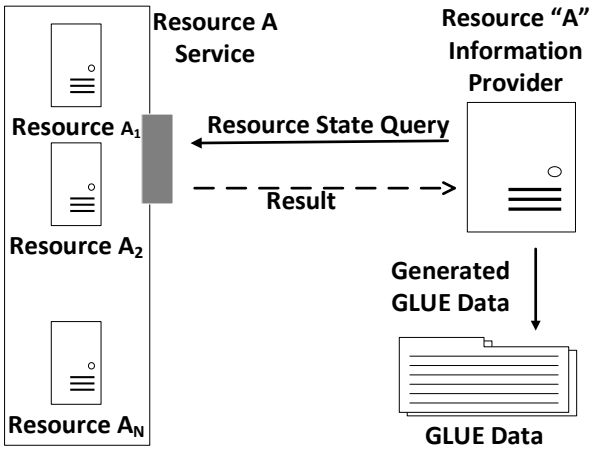


Figure 2.12: The abstract Grid Information Provider model. An Information Provider queries a service that manages a set of Resources of type "A" at a resource centre. The Information Provider transforms the response into one or more GLUE entities.

tion System follows a natural hierarchical structure (Figure 2.13): GLUE entities are generated by Information Providers; the set of Information Providers (info-providers) on a particular node publish their state as a *local resource*; the set of local resources form a *domain*; and the combined set of entities from each domain yield a global view.

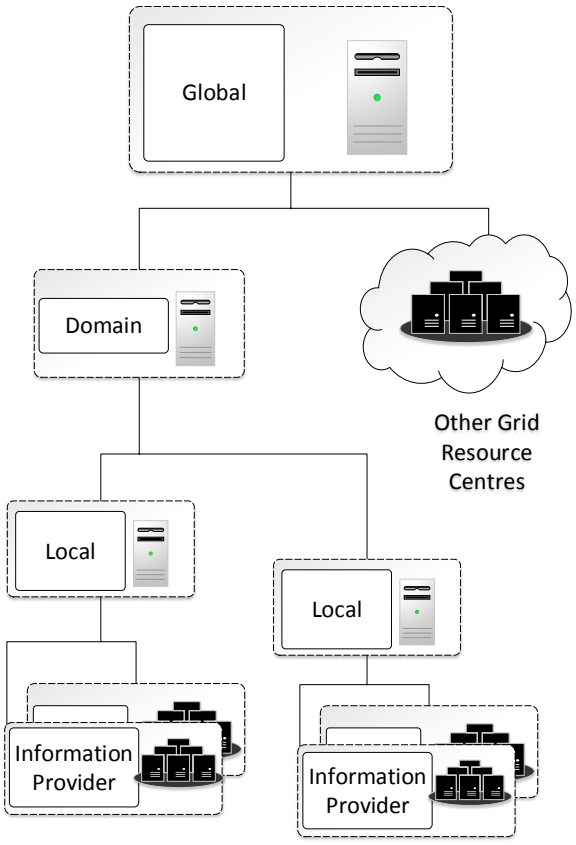


Figure 2.13: The Grid Information System. GLUE data is propagated from the lowest level (generated by Information Providers) up to Global level.

The GLUE information "presentation layer" is typically managed by the *BDII*[15], and the presentation of GLUE in a concrete format such as LDAP or XML is known as a *rendering* (a conceptual model is independent of any rendering). This is an implementation of a hierarchical Grid Information System model with three BDII "types" and a set of "information providers" that generate information about the Grid entities. As per Figure 2.13, the Resource-BDII (lowest BDII level) aggregates the state of a grid service node by executing a set of Generic Information Providers (GIP) plugins; the Site-BDII aggregates all the Resource-BDIIs belonging to the given site; and the Top-level BDII aggregates all the Site-BDIIs. Information is "pulled" from the lower to higher levels. In general, all queries about the state of the Grid are made through the Top-Level BDII.

The LDAP rendering is the most widely used Grid Information System, having become the dominant solution due to the need for grid inter-operation. However, it is only one of several methods that may be used to publish GLUE information. Other renderings that have been used include XML (the Nordugrid ARC middleware [108]) and SQL (R-GMA [109]).

2.4.2 Integrating New Hardware into a Grid

Widening the range and capabilities of Grid infrastructures to handle additional hardware, such as sensor-based instruments and computational accelerators, has been examined by several other researchers and project teams.

For example, early work (2004) looked into integrating FPGAs into a Globus-based grid using the Proteus Software Platform [110]. This work exposed the FPGA as a new grid service called the *ProteusGridService*. Similarly, more recent work uses FPGAs to accelerate profile Hidden Markov Model (HMM) searches of biological sequence databases on a small grid-enabled cluster [111]. However, it is not clear from these papers as to whether the services were advertised in the grid information system, or whether their usage depended upon a priori knowledge to access them.

GridCC [112], RinGrid [113] and DORII [114] were a series of projects concerned with providing grid-based access to remotely operated and controlled scientific instruments and sensors. The *InstrumentElement* was initially developed within the GridCC project as a grid interface to distributed instrument monitoring and control through a web service using grid security mechanisms. The RinGrid project's core objectives

were to increase the utilisation of instruments and sensors by using the grid protocols to manage resource sharing, and to allow the data produced by the instruments to be stored and processed on the Grid itself. Furthermore, workflows (i.e. the ability to execute a chain of related applications) using multiple instruments were supported. The DORII project followed on from RinGrid and was aimed at extending the range of instruments that could be managed on the Grid. New grid-enabled applications included synchrotron-based experiments in X-ray imaging for material sciences and medicine [115] and ocenographic modelling [116]. The DORII project produced an update to the GLUE 1.3 schema that allowed Instrument Elements to publish service and access control information through the BDII grid information system. The key aspects of this work are that: (i) instruments and their web service interfaces were discoverable; and (ii) user access to instruments was through a web-based user interface called the Virtual Control Room. However, standard grid job submission through the CE to an LRMS was not suitable in this case – as some instruments could require real-time interaction and might not be amenable to batch processing.

In 2008 the IBM Roadrunner became the worlds fastest supercomputer, breaking the one Petaflop per second barrier. It was the first supercomputer to be based on a hybrid architecture of AMD x86-64 compatible processors and the Cell PowerXCell8i, a PowerPC-based general purpose processor with eight coprocessors [117]. The Cell was also used in the Sony PlayStation 3 (PS3) games console, a system that was also capable of running the Linux operating system. The low cost of the PlayStation (approximately $350) made it possible to build budget "supercomputers" that were benchmarked to execute some applications up to thirty-three times faster than a contemporary single core Intel-based processor, and six times faster than a contemporary quad-core processor [50].

In work aimed at increasing the range of processor architectures and systems running the gLite middleware beyond Intel-based x86 systems, the Grid-Ireland (see Page 65) operations team worked on increasing the portability of the gLite worker node software stack. Initial targets included the SGI Irix MIPS/RiSC-based OS and Apple's PowerPC-based X-serve running OS X. This work was further developed to support the PS3 Cell-based systems [118]. One goal of this latter work was to support the eHITs [119] life-sciences application, an application that was optimised for execution on the Cell processor. One of the weaknesses of the integration of the PS3 into the grid was in how the resources were advertised on the grid – namely, the set

of PS3 nodes were partitioned in the LRMS to be accessed through a specific queue. No information about the capabilities of the resources were advertised other than the system type – the tag "ps3" was hard-coded into the name of an LRMS queue, and advertised on the grid as a CE-endpoint. This tag conveyed no meaningful information to users or other grid services, and to effectively use the PS3 resources, the user was expected to have a priori knowledge about the association between the queue name and the resource. To restrict usage to a set of "power" users, access controls based on VO membership were also applied. In addition, the total number of processors advertised in the grid information system for that CE did not distinguish between Intel-based and Power-PC/Cell based processors. Although it would have been possible to grid-enable the PS3 by using a separate CE at the site, there were additional overheads associated with integrating and managing extras CEs on the grid, and there were limited ways to advertise additional architectural information within the Glue 1.X schema.

A similar queue-based approach was taken by Grid-Ireland to integrate a collection of hybrid 8-core x86-64 processors/Nvidia GPGPUs into their infrastructure. The GLUE 1.3 schema had limited scope to allow both the CPU and the GPGPU capacity to be advertised (this limitation is discussed further in Chapter 3.2). A naïve strategy that publishes the number of GPGPUs as the compute capacity will result in under utilisation of CPUs.

Publishing appropriate GLUE information is a vital part of how grids work. However, based on the experience gained from the aforementioned examples: (i) deriving a new GLUE type (e.g. Instrument Element) may take several years; and (ii) the Grid-Ireland PS3 and GPGPU integration experience has shown that it is not sufficient to publish merely the existence of a resource, as the lack of semantic detail hinders discovering real differences between available resources, and there is no satisfactory way to publish capacity and utilisation information that reflects true resource usage of the CPU and GPGPU.

2.4.3 The EGI GPGPU Surveys

The EGI is a federated grid infrastructure consisting of over 310 resource centres spread across more than 36 European countries (National Grid Infrastructures). This Grid supports a user-community of more than 22,000 researchers from a diverse range of scientific backgrounds. The current compute service provides more than 485,000 logical

CPUs [120]. The EGI evolved out of a series of successful European Commission funded grid projects (including EDG, EGEE I-III, CrossGrid and Int.EU.Grid). Its principal aim is to deliver a unified, sustainable, integrated and secure grid computing service for European researchers.

In June 2012 an EGI Working Group was established to gauge interest in GPGPU computing services on the Grid[1]. The working group produced two surveys, the first aimed at the grid resource providers, and the second aimed at e-Science researchers (grid and non-grid users). The surveys were released in August 2012 and were disseminated through the national grid infrastructures and through the EGI's own user-community support channels. The surveys closed in September 2012. The surveys were the first attempt to gauge the level of interest for GPGPU-based computing within the grid community.

The Resource Centre Survey

The resource centre survey consisted of thirteen questions, and its goals were to: (i) gauge whether GPGPUs were already being used on the grid infrastructure or to determine whether resource centres planned to deploy them; and to (ii) build up shared community knowledge about GPGPU resource integration into existing LRMSs, the GPGPU allocation policies in use, and how these GPGPUs were advertised on the grid (if they were indeed advertised at all).

A total of 44 responses were received. Furthermore, some of those responses were on behalf of NGIs, who replied on behalf of several constituent resource centres. The questions were non-mandatory, so the questions did not solicit an equal number of responses.

Q1 Does your resource centre currently provide GPGPU resources?

43 Responses: Yes 13 (30.2%), No 30 (69.8%).

Q2 Do you plan to further extend the amount of GPGPU compute capacity offered in the coming 24 months?

Based on 13 "Yes" Responses from Q1: Yes 11 (84.6%) No 2 (15.4%).

Q3 Will your site provide GPGPU resources in the coming 24 months?

41 responses: Yes 23 (56.3%) No 18 (43.9%).

[1] I was a leading member of this working group and wrote the proposal to establish it.

Q4 Which LRMS is, or will be, used to provide access to GPGPUs?

22 Responses: TORQUE 18, Slurm 6 (multiple choice answer) .

Q6 Does/will every node on your cluster have a GPU?

Yes (21.7%) No (78.3%).

Q9 Please provide information about the GPGPU hardware deployed in your site.

13 Responses, typical configuration indicated mostly 1 or 2 GPGPUs per host, but a couple of responses stated 4 and 8 GPGPU devices per physical machine.

A number of important conclusions can be drawn from the survey results. While a significant number of resource centres have already deployed GPGPUs, many more resource centres planned on doing so over the coming years. There is a clear case for the development of a solution that will support GPGPU discovery and job specification support in the grid middleware; most resource centres planned to deploy GPGPUs on a fraction of their overall clusters, rather than uniformly across entire clusters, with variations in the number of GPGPUs per worker node expected (potentially even within the same cluster). This result has important implications for the management of GPGPU resources. Finally, most GPGPU deployments were expected to be within clusters using the TORQUE LRMS, i.e. an LRMS with minimal support for GPGPUs.

The User Survey

The User survey was designed to gauge the level of interest in using GPGPUs on a grid infrastructure, to examine who was already using them (both grid and non-grid users), and how they were being used. There were 23 non-mandatory questions in total, and 48 people responded. The results for some selected questions are listed below.

Q1 Do you currently use grid or cloud technologies?

44 responses: Yes 37 (84.1%), No 7 (15.9%).

Q2 Would you be interested in accessing remote GPGPU based resources through a computing infrastructure, such as National Grid Infrastructures or the European Grid Infrastructure (EGI)?

42 responses: Yes 39 (92.9%), No 3 (7.1%)

Q3 Do you use GPGPU based applications for your scientific computations?

47 responses: Yes 30 (63.8%), No 17 (36.2%).

Q8 Do you develop or intend to develop any GPGPU based applications?

41 responses: Yes 22 (53.7%), No 19 (46.3%).

Q13 What Application Programming Interface do/will you use?

17 responses with multiple selections possible: CUDA (94.1%), OpenCL (41.2%), OpenACC (17.6%), Other (17.6%).

Q14 Do you intend to develop code which depends on other application frameworks (e.g. MPI, BLAST)?

13 Responses with multiple selections possible: MPI (100%), BLAST (15.4%), Other (7.2%).

Q18 What is the optimal ratio between the number of GPGPUs and CPU cores for your application?

12 responses: 5 Don't know or misunderstood the question (mixing GPGPU cores for CPU cores), 1 person indicated a 1-to-1 ratio, and 6 indicated that the ratio was not 1-to-one with either more CPUs than GPGPUs, or more GPGPUs than CPUs.

The survey showed that from those who responded, there was over 90% support for having access to GPGPUs on the grid. This result in itself will have inherent bias, as it more likely that those who responded had an interest in the topic to begin with. However, some key points that should not be influenced by this stand out, namely: (i) there is a user demand for GPGPU support on grids; (ii) the CUDA, OpenCL and other APIs are needed for the user applications; (iii) some users expected to use additional APIs, such as MPI, with their application; (iv) it could not be assumed that user applications would require 1 CPU to 1 GPGPU resource allocation. Support for other allocation patterns where there are more CPUs than GPGPUs, and vice-versa, are needed.

2.4.4 Survey of the EGI Grid Information

Section 2.4 introduced the basics of Grids and grid computing. However, as shown in Table 2.6, there is a wide range of grid middleware available for resource providers to choose from when connecting a cluster to a grid. Since Chapter 3 and Chapter 4 both address the integration of new resources into a grid, it is worth capturing some statistics about the current state of middleware deployment. These statistics will be used to

determine where effort spent in the development of resource integration solutions may be best placed.

The data available through a Grid Information System provides a wealth of information about the state of a Grid. The Top-Level BDII chosen for this analysis was *lcg-bdii.cern.ch*. This particular Grid Information System collates information about the combined set of resources made available to the CERN-based WLCG Grid. The WLCG includes resources from both the EGI and the OSG. Furthermore it is a *non-homogeneous* grid infrastructure using a wide range of grid middleware and resource management systems. The snapshot, which was taken in October 2015, shows that there were a total of 349 Sites and 567 unique ComputeElements deployed. In addition, the number of ComputeElement queues published was 7226.

To determine what middleware and LRMSs are commonly used on the ComputeElements, the GLUE 1.3 GlueCEUniqueID, GlueCEImplementationName and GlueCEInfoLRMSType values were extracted from the snapshot. A summary of this data is shown in Table 2.7. Furthermore, the cumulative graphs (Figure 2.14 and Figure 2.15)

	AliEN	**ARC-CE**	**CREAM**	**Globus**	**HTCondorCE**	**Total**
Condor	0	29	9	20	22	**80**
(S)GE	0	6	43	3	0	**52**
LoadL.	0	2	0	0	0	**2**
LSF	0	0	69	2	1	**72**
PBSPro	0	1	2	0	1	**4**
SLURM	1	26	12	1	1	**41**
TORQUE	0	12	289	11	4	**316**
Total	**1**	**76**	**424**	**37**	**29**	**567**

Table 2.7: Grid ComputeElement middleware deployments and their LRMS type

of the data from Table 2.7 clearly indicate that the CREAM ComputeElement implementation (a product of the UMD middleware) is the by far the most popular ComputeElement used (427 of 567 instances, or 75%). Furthermore, TORQUE is the most popular choice of LRMS, and is deployed on 316 of 567 ComputeElements (55.7%). In the case of CREAM-based ComputeElements, it is deployed on almost 51% of all instances.

Finding GPGPU Needles in a GIS Haystack

In order to establish if GPGPUs were advertised on the Grid, a number of Grid Information Service searches were carried out. For example a case-insensitive search of the

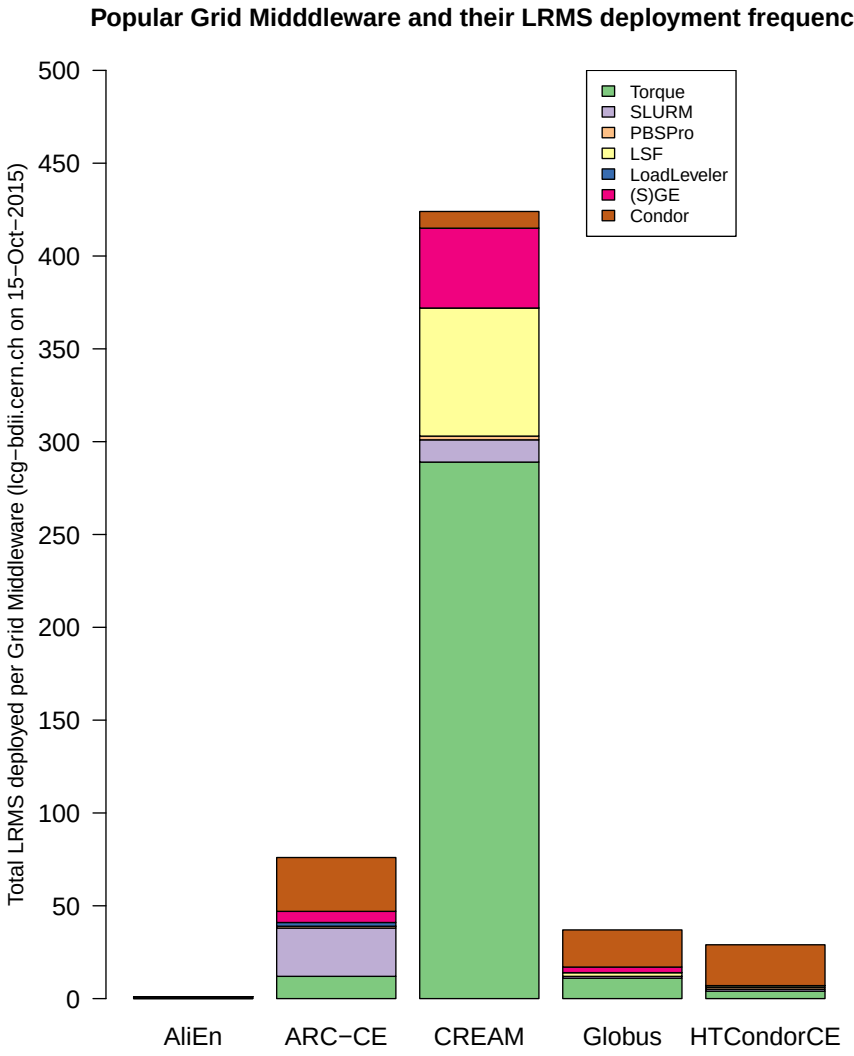


Figure 2.14: The frequency of Local Resource Management Systems installed on grid Compute Elements in relation to type of underlying grid middleware. Figures were derived from data published on the lcg-bdii.cern.ch Top-level BDII (15-Oct-2015)

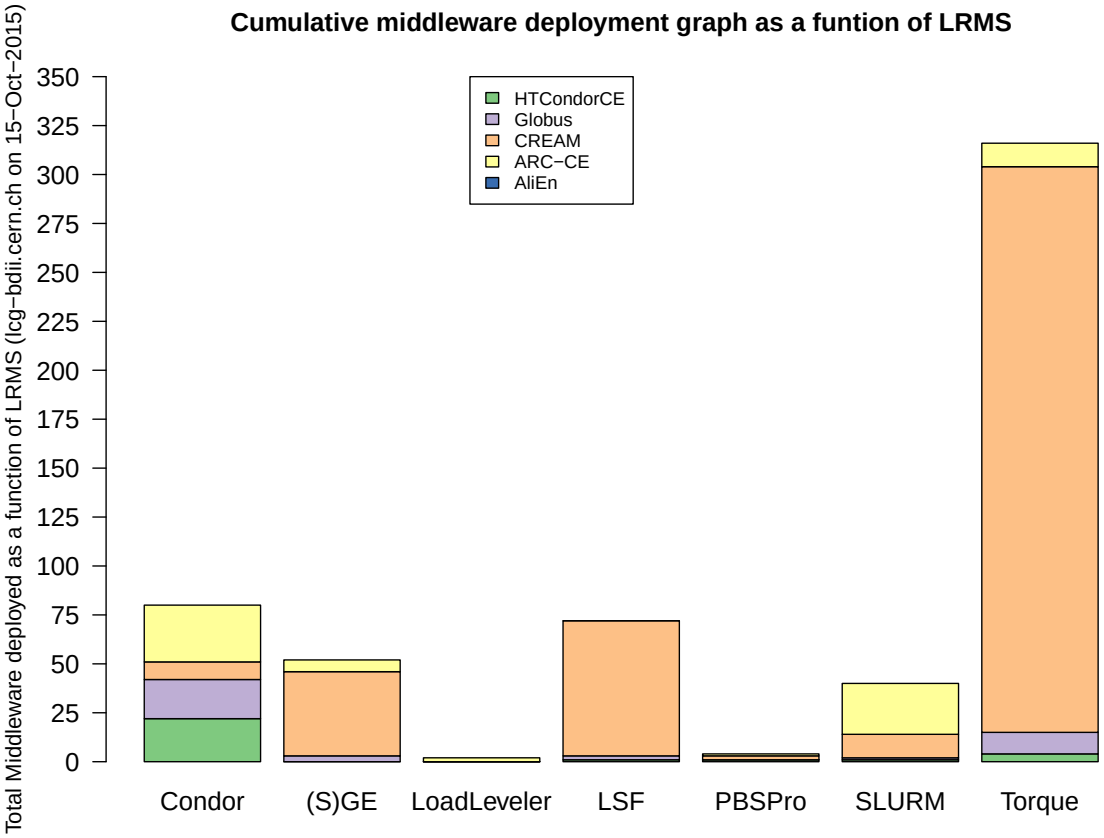


Figure 2.15: The cumulative frequency of grid middleware used by ComputeElements relative to the LRMS. Figures were derived from data published on the lcg-bdii.cern.ch Top-level BDII (15-Oct-2015)

Listing 2.7: A list of ComputeElements using the name of a queue to indicate GPGPU deployment. The results show only Compute Element queue endpoints matching the term "GPU"

```
ce01.cr.cnaf.infn.it:8443/cream-lsf-gpu
ce01-lcg.cr.cnaf.infn.it:8443/cream-lsf-gpu
ce01.tier2.hep.manchester.ac.uk:8443/cream-pbs-gpu
ce02.tier2.hep.manchester.ac.uk:8443/cream-pbs-gpu
ce03.tier2.hep.manchester.ac.uk:8443/cream-pbs-gpu
ce04-lcg.cr.cnaf.infn.it:8443/cream-lsf-gpu
ce05-lcg.cr.cnaf.infn.it:8443/cream-lsf-gpu
ce06-lcg.cr.cnaf.infn.it:8443/cream-lsf-gpu
ce07-lcg.cr.cnaf.infn.it:8443/cream-lsf-gpu
ce08-lcg.cr.cnaf.infn.it:8443/cream-lsf-gpu
ce1.accre.vanderbilt.edu:2119/jobmanager-slurm-gpu
ce2.accre.vanderbilt.edu:2119/jobmanager-slurm-gpu
kodiak-ce.baylor.edu:2119/jobmanager-pbs-ecsgpu
vm3.tier2.hep.manchester.ac.uk:8443/cream-pbs-gpu
```

GIS for the term "gpu" found:

(i) a small number (14) of Compute Elements publishing data matching that term in the name of a queue (see Listing 2.7);

(ii) two resource providers choosing to use a GLUE 1.3 SoftwareEnvironment tag to publish the presence of GPGPU resources (Listing 2.8);

(iii) no other information containing the term "gpu" term, implying no GPGPU property or state information was published.

Similar searches for other terms such as "CUDA" and "OpenCL" also yielded zero matches.

Listing 2.8: Deployment of GPGPUs indicated by publishing a simple GLUE Application Tag

```
dn: GlueSubClusterUniqueID=aesyle-grid.fgi.csc.fi
GlueHostApplicationSoftwareRunTimeEnvironment: ENV/GPGPU-FERMI

dn: GlueSubClusterUniqueID=cream01.grid.auth.gr
GlueHostApplicationSoftwareRunTimeEnvironment: GPU
```

It can be inferred from the information examined in this survey that GPGPUs are already being deployed on grid infrastructures using a variety of grid middleware and using a variety of different LRMS types. For example, the deployment of GPGPUs on Compute Elements using UMD and Globus is implied from the format of the Compute Element values in Listing 2.7, and the use of LSF, PBS/TORQUE and Slurm LRMSs can also be inferred from Listing 2.7.

The lack of a consistent scheme to publish even minimal information about the presence of GPGPUs is also evident. For example, the two compute elements shown in Listing 2.8 do not appear in Listing 2.7. Furthermore, no information is published about the capabilities of different GPGPUs and their utilisation, so users cannot make an informed decision about which resource provider's GPGPUs are the most suitable ones to select for their jobs.

The Grid-Ireland Legacy

Grid-Ireland [121] was established in October 1999 as a pilot project between three geographically dispersed universities – Trinity College Dublin (TCD), University College Cork and the National University of Ireland, Galway. The project initially used the Globus middleware, which allowed authenticated and authorised users to access the grid-enabled computational resources within each of the institutions.

During the 1999 to 2003 period, the project became closely aligned to European Commission funded grid projects, and the deployment of updated grid middleware followed suit. This included the European Data Grid (EDG) and CrossGrid (as a project partner). The EDG middleware included a new range of group-management, security and data-oriented services that enhanced the existing Globus-based computational services, while CrossGrid developed compatible grid middleware and applications that supported user interaction.

In 2001 the Cosmogrid project, a consortium that would help establish a prototype cross-border (island of Ireland) computing grid service, was proposed. The project proposal was successful and started in 2002. The consortium included the additional academic institutions of the Dublin Institute for Advanced Studies, Dublin City University, University College Dublin, as well as Mét Eireann, Armagh Observatory and the national research and education network (NREN) HEAnet.

The Cosmogrid project introduced some novel approaches to the roll-out of grid services, namely: (i) it established a centralised Grid-Ireland Operations Centre in TCD; (ii) the Operations centre remotely managed all grid services; (iii) the institutions (sites) were provided with "Grid Gateways"; (iv) institute cluster resources were then connected to the Grid Gateways; (v) the Grid Gateways used machine virtualisation extensively [122] [123]; (vi) a virtualised replica (TestGrid) of the deployed infrastructure was used to validate changes to the grid middleware and services [122]; (vii) it supported the transactional deployment of validated operating system and middleware updates to all sites [124]; (viii) grid service monitoring was key to the centralisation.

Two of the key Grid-Ireland objectives were driven by the needs of the Irish grid community, namely: (i) MPI-based applications had to be supported on the Grid; and (ii) the grid middleware had to be supported on a wide range of cluster systems. As a result, the Grid-Ireland Operations Centre staff were able to bring experience and expertise with grid-enabled MPI application support, middleware portability and virtualisation into a number of European Commission funded projects (EGEE-I, EGEE-II, EGEE-III, EGI-InSPIRE, Int.EU.Grid, StratusLab, Mantychore, HELIO and SCI-BUS) and Irish government funded projects (Webcom-G and e-INIS). During this time the Operations Centre were also early adopters of provisioning for the extension of grid information systems [125] [126] and the use of accelerated computing technologies such as the Cell-based PS3 cluster and GPGPUs for computations and visualisation [127].

Grid-Ireland was also a founding member of the European Grid Infrastructure (EGI) in 2010. This established a legal foundation and council to oversee a pan-European grid infrastructure based around cooperating national grid infrastructures (NGIs). Each NGI would operate a national service to monitor the availability and reliability of resource providers, run grid help desks, investigate issues with resource providers, and set out procedures for resource integration and daily operations. The NGI's role was akin to (and a subset of) the operations model that the Grid-Ireland Operations Centre had used since 2003.

Grid-Ireland ceased grid operations in December 2012 as a consequence of the financial crisis, by which time it supported over 25 virtual organisations and had provided over 70 million normalised (HEP)SPEC06 CPU hours to the LHC Computing Grid.

2.5 Other Related Work

GPU resources on the Grid

There is existing prior art where GPGPU resources have been partially integrated into Grids. These approaches consider both non-virtualised GPGPU resources [128] and virtualised GPGPU resources [72]. In both cases, the grid user's job is given exclusive access to the GPGPU resource. However, both of these implementations lack resource and service discovery. The user must have *a priori* knowledge of the existence of the resources, and must know how to request those resources (which is resource provider specific). The work presented in this thesis provides resource discovery and includes support for job submission.

BOINC and Desktop Grids

BOINC has support for multi-disciplinary computational sciences using GPGPU. For example, the *Einstein@Home* project uses GPGPUs to search for weak astrophysical signals from spinning neutron stars (also called pulsars) using data from the LIGO gravitational-wave detectors, the Arecibo radio telescope, and the Fermi gamma-ray satellite [129]. Furthermore, the EDGI Desktop Grid provides a mechanism [130] to bridge between EGI and BOINC-enabled resources. However, such combinations do not address the specific GPGPU service-discovery requirements.

HTCondor GPGPU Support

HTCondor [23] supports advanced GPGPU resource publication, service-discovery, per-job GPGPU match-making, and job-management [131]. Indeed, HTCondor is central to the the UMD WMS match-making service. However, some of the major differences between HTCondor and the work presented in this thesis include: (i) the solution is intended to be compatible with Grids using the OGF GLUE 2.0 information model; and (ii) the approach taken treats GPGPUs as instances of consumable resources.

EGI GPGPU Support

The European Grid Initiative (EGI) started to explore the question of GPGPU resource integration, which is only one of the types of resources that this thesis aims to integrate

into the Grid.

The thesis author was a founding and leading member of the EGI GPGPU Working Group [132], having presented several talks and hosted EGI project workshops on the GPGPU integration problem. The author also wrote the proposal that led to the establishment of that working group. The initial goals of the working group were to examine how progress on the GPGPU integration could be made.

The goals of the working group were subsumed by a new activity in the EGI-Engage project in March 2015. The roadmap for this activity sets out key goals which include the integration for GPGPUs into the EGI Grid, and the inclusion of access to GPGPUs as a user service in the EGI Federated Cloud.

There are several key differences in the work presented in this thesis, and the work carried out the new activity. The EGI activity is proposing a change to the GLUE Schema that will only support GPGPUs. This change will also need further changes to the grid middleware, and is contrary to the objectives of this thesis (i.e using existing standards and making small changes to the core grid middleware). The sub-activity that is investigating GPGPU virtualisation are limiting their investigation to PCI-Passthrough, rather than looking at more generalised solutions such as OS-Level Virtualisation.

The concepts, architectures and results presented in this thesis represent distinct independent work that may be contributed back to this community in the future.

EGI Federated Cloud Working Group

The EGI Federated Cloud Working Group uses the GLUE 2.0 ExecutionEnvironment to advertise diverse sets of (virtual) CPU-related resources [133]. I had considered using this approach, but deemed it to be unsuitable as a way to describe *Consumable Resources*. The ExecutionEnvironment describes properties relating to the a set of worker nodes, but not the additional resources that they might control.

2.6 Summary

The importance of parallel computing as a solution to the problem of ever increasing demands placed on computational processing is widely accepted. As is evident from the different abstract parallel computing architectures (in particular, SIMD and

MIMD), many approaches to parallelisation (including hybrid SIMD/MIMD) are possible. These different approaches give rise to very distinct hardware, compiler and API solutions. The most typical parallel computing solutions are: (i) augmenting the CPU with vector operation capabilities (SIMD); (ii) increasing the number of CPU cores on a single physical CPU, each capable of working on distinct data (Multi-core MIMD); (iii) allowing large numbers of CPUs on different computers to communicate with one another over a network (e.g. MPI); (iv) using additional accelerator hardware that supports massively parallel lightweight computations; or (v) using combinations of the these approaches. The bi-annual Top500 HPC lists show an increasing dependency on accelerators as a solution for handling increasingly complex user applications.

Grids are complex distributed systems of clusters that rely on standards (e.g. information publication, security) to provide services to their users. Changes to these standards are non-trivial. For example, a change in how information is published (on the Grid) may need corresponding changes to many other grid middleware components or services. The changes need to deal with the complete chain of job orchestration services (i.e. resource discovery, job submission on the UI, brokering/workload management, LRMS integration, dynamic information provision). Indeed, previous experience with developing a solution to support multi-CPU/multi-core parallel applications using MPI or OpenMP has shown that new services may require several years to deploy. Furthermore, changes need to be carefully tested to ensure that they have no impact on the stability of a production service.

The emergence of several different types of massively-parallel accelerators (such as GPGPUs, Intel's Xeon Phi and, to a lesser extent, FPGAs), introduces a new set of grid integration challenges in addition to those noted above, namely: (i) accelerators do not have a uniform set of properties; and (ii) LRMS support varies from accelerator to accelerator, and that support varies even for the same type of accelerator. e.g. Nvidia vs AMD GPGPUs. It is clear that this diversity compounds an already complex (grid and non-grid) resource integration problem.

The EGI Resource Centre and User Surveys are clear indicators that both grid resource providers and users would like to see (GPGPU) accelerators integrated into the grid. The Resource Centre Survey gives insight into the LRMS and physical configuration of computers hosting GPGPUs (multi-GPGPU configurations in many cases), while the User Survey gives insight into the type of applications, APIs and other user requirements (e.g. exclusive GPGPU resource access, multi-GPGPU allocations) that

a GPGPU-enabled grid infrastructure should be capable of handling. Although there has been prior work that attempts to integrate GPGPUs into grid infrastructures, these efforts do not publish any information about the GPGPU resources (so the grid user must know where the resources are located), or are limited to particular middleware and have very limited resource matching capabilities. Furthermore, the approaches in the past taken do not consider other types of accelerators, so they are limited in how they can adapt to future changes. There is a clear opportunity to develop a grid solution that accommodates a greater range of grid middlewares and accelerators.

As with GPGPUs and accelerators, the emergence of machine virtualisation and Cloud Computing is already causing a profound change in the manner in which computing services are provided to users. In particular, Grids such as EGI are moving towards a Federated Cloud infrastructure that permits the user to execute applications within virtual machines that are specifically tailored to the application needs (rather than the conventional approach that requires the application to fit into a given static operating system environment). However, designing and deploying virtual machines with virtualised accelerators is a much more difficult task. Low-level efforts to integrate virtual GPGPUs using API-interception have been bound to specific LRMSs and bound further still to a specific API (e.g. SLURM and VCL). There does not seem to be an existing solution that can be used with many LRMSs and that can support multiple virtual GPGPUs using API-interception. As with the grid/accelerator integration challenge previously highlighted, there is an additional opportunity to develop a virtual GPGPU grid solution that accommodates multiple grid middlewares, and a wider range of accelerators and virtualisation technologies.

Chapter 3

CPU-Dependent Execution Resources in Grid Infrastructures

3.1 Introduction

This thesis argues that previous attempts to *integrate* (p.2) new types of computational resources, such as computational accelerators or FPGAs, into Grids (e.g. EGI and OSG) did not sufficiently address the need to exploit complex user-driven resource discovery or specification support. These integration attempts had narrowly focussed on individual types of resources rather than developing a more general solution to the resource integration problem. This argument is supported by the evidence of previous attempts to integrate new computational accelerators ([118],[128],[72]) and grid instruments ([112],[113],[114]), as well as the time it took to implement and deploy MPI/OpenMP parallel computing support (noted in Chapter 2). The notable pitfalls of these solutions were typically:

(i) the focus was solely on GPGPUs, and not other resources ([128],[72]);

(ii) there was no resource advertisement. Consequently this required the grid user to have prior knowledge about how (and where) to target these resources and execute jobs on them ([118],[128],[72]);

(iii) the solution was tied to specific middleware implementations, and lacked the ability to handle complex resource requirement expressions ([118],[128],[72]); or

(iv) the solution required changes to both the GLUE standards and the middleware. This had direct consequences on the ability to rapidly implement support for new

grid resources (see the remark about GLUE 2.0 ratification in 2009 on p.52 and [112],[113],[114]).

One of the challenges facing Grids is how to proceed with the integration of a diverse range of new computational resources into existing grid infrastructures. If the grid infrastructure cannot deal with resource integration in a timely manner, then the user-communities may look for solutions outside of the Grid (e.g on commercial Cloud-computing infrastructures such as Amazon's AWS [134]). However, by rushing through major changes to grid middleware and their components, there is a risk of introducing errors and widespread instabilities. Making rash changes to grid middleware is contrary to the goals of supporting long-term stable infrastructures. Grid infrastructures need to provide high levels of service availability and reliability [135]. For example, EGI stipulates that participating resource providers must meet monthly targets of 80% service availability and 85% service reliability [136].

An objective of the work presented in this thesis is to simplify and accelerate the procedure for integrating new computational resources into grid infrastructures, but without making changes to the technological standards used by Grids. It is therefore necessary to minimise, or better avoid, changes to the core of the grid middleware. An underlying philosophy of this thesis is to mitigate against introducing new software bugs and other sources of grid middleware or operational instability. Any new resources must be integrated into existing middleware without the need to build new and complex grid services. Furthermore, rather than concentrating on integrating a single type of resource (e.g. GPGPUs), an abstract approach to resource integration is taken that: (i) examines the broader question of what strategies are available to integrate a wide range of computational resources into a grid infrastructure, (ii) investigates ways to enable new resource types on the grid that are discoverable and accessible in a manner that is almost on-par with standard CPU resources, and (iii) applies those strategies to new computational resources on the infrastructure.

In Section 3.2, an abstract model that enables a flexible, dynamic approach to the integration of new (and yet-to-be-conceived) resources into Grids is proposed. The abstract model provides the necessary level of abstraction required to treat the integration of many types of computational resources in a uniform manner. To facilitate this, a resource classification called the *CPU-Dependent Execution Resource* (CDER) is proposed. The CDER is a type of consumable resource that can execute its own in-

structions (e.g. a computational accelerator) and has non-trivial properties and states (e.g. performance, architecture, memory capacity) in its own right. While the CDER classification may be less useful in a cluster-only environment, it is of significant benefit in grid environments. The publication of the different CDER properties and states could allow a grid user to discover and compare similar CDERs with one another. For example, a grid user may want to locate resource providers with certain FPGA resources that match a particular hardware profile.

Based on a review of how grid information is currently published using the existing GLUE Schema standards, Section 3.2.1 proposes and describes five conceptual strategies for discovering and accessing CDERs. The two most favourable strategies address the publication and discovery of CDER data, but cannot be used with current grid job submission applications. A new two-phase job submission mechanism is proposed in Section 3.2.2 to overcome this limitation. This mechanism supports CDER resource matching and job submission through a WMS. These strategies are evaluated against a number of criteria in Section 3.2.3 and conclusions are drawn about the most appropriate strategies for CDER integration.

A proof-of-concept application of the conceptual approach to GPGPU CDERs is described in Section 3.3. It will be shown how to integrate GPGPUs into the Grid with full resource discovery and job specification support.

3.2 Abstract Approach to Resource Integration

The thesis proposes an abstract approach to the integration of new worker node resources, such as GPGPUs, into any LRMS. To achieve this, the approach for handling the integration of many types of resources into LRMSs and Grids (e.g by examining common traits and requirements) is studied and formalised. However, regardless of whether a job requires a particular property or resource, it is still dependent on the allocation of a CPU to execute.

Definition: A *CPU-Dependent Execution Resource* (CDER) is defined here to be a computational resource that is characterised by the following properties:

(i) access to the resource requires a CPU to be allocated by the LRMS,

(ii) there are a limited number of resources,

(iii) each resource is bound to a specific worker node,

(iv) the user that has been allocated the resource perceives they have exclusive access to it,

(v) the resource has its own set of physical characteristics and measurable properties (e.g. memory capacity, performance, hardware characteristics).

The resources do not operate independently of the worker nodes, and operate by leveraging one or more CPUs. The requirement that the quantity of CDERs are finite implies that the acquisition and release of these resources must be adequately managed and accounted for by the LRMS or some other external service. Furthermore, when a CDER is allocated to a job, it may not be allocated to another job until the current job releases it. Allocation must be controlled through some form of queue or scheduler (c.f other resources such as Input/Output systems that are managed by an Operating System). This restriction is intended to help manage resource allocation, and to avoid more than one job simultaneously accessing the same resource. A danger of not enforcing this requirement may be longer job execution times. This reinforces the requirement that the job has exclusive access to the CDER.

The CDER definition is intended to encompass a range of resources such as GPGPUs, FPGAs, and other hardware devices that are physically attached to worker nodes (Figure 3.1). The definition excludes software applications (which already have a GLUE definition). Furthermore, the problem of integrating software that is "node-locked" due to licensing restrictions (i.e. the software can only be executed on particular worker nodes) is similar to the CDER integration problem. However, unlike the CDERs, node-locked software doesn't have any interesting characteristics or state information (other than utilisation) that are relevant when specifying the resources required by a job.

The GPGPU as a CDER

Do GPGPUs conform to the definition of a CDER? Examining the conditions it can be seen that: (i) the GPGPU can only be used in conjunction with a CPU; (ii) the number of GPGPUs installed on a machine is physically limited; (iii) the GPGPU is physically attached/bound to the machine; (iv) exclusive access is not necessarily guaranteed by every LRMS (Section 3.3.2), but it can be managed (Section 3.3.2);

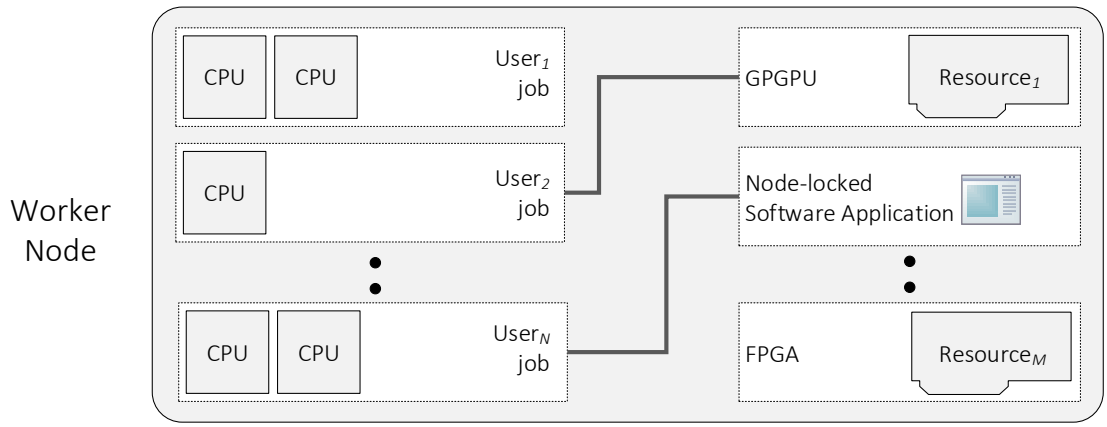


Figure 3.1: An example worker node with several CDERs (a GPGPU and an FPGA). The LRMS is responsible for assigning CPUs to each job, and each CDER resource can only be used by one and only one job at any one time.

(v) the GPGPU has interesting properties that may influence resource selection (e.g. performance, architecture, memory capacity).

One of the user requirements from the GPGPU User survey was that the GPGPU should be exclusively dedicated to the user's job, and *should not* be treated as a resource that is concurrently shared with other users. Enforcing exclusive allocation ensures that distinct user jobs cannot inadvertently exhaust the GPGPU's resources (e.g. GPGPU memory) or cause contention for computational time. So as long as guaranteed controlled access to the resource is enforced, the GPGPU can be regarded as a CDER.

CDERs as First-Class Grid Resources

The CPU core is one of the most fundamental resources available on a Grid (one other major resource type is disk storage, but its treatment is outside the scope of this thesis). A computational grid infrastructure is, by design, built around the idea of being able to pool together and securely share a set of distributed CPU resources. Through the GLUE schema and the Grid Information System, grid users have a consistent and uniform way in which to find CPU resources and to determine their properties or usage – this exemplifies *Discovery*. When a grid user specifies the number of CPU resources required by their job, that specification must be translated into an equivalent LRMS request by the target Compute Element. Despite the differences in how CPU resources are managed internally by each LRMS, there is also a common consistent external

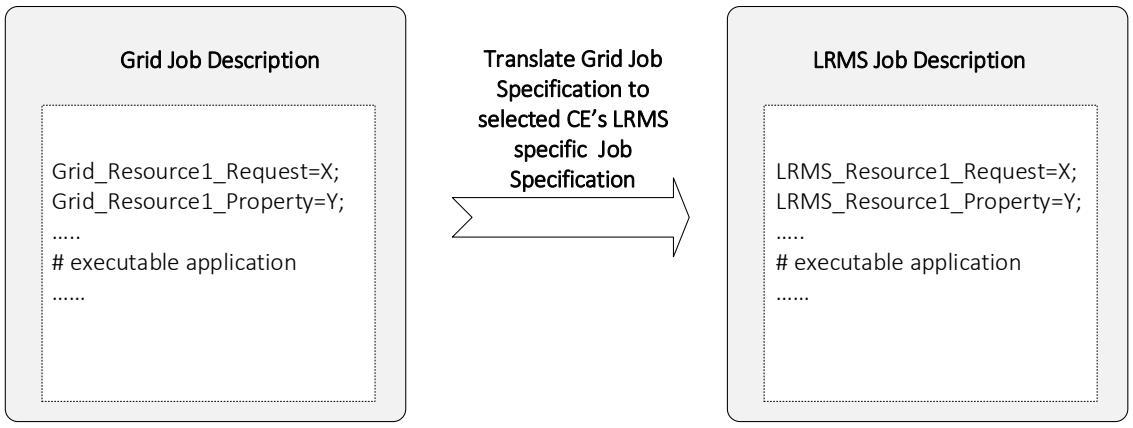


Figure 3.2: A resource specification must be translated into an LRMS specific resource specification.

representation that allows the user to submit a job to any of the resource centres – this property exhibits *Independence* from the LRMS. The same principle must apply to any other resource type (Figure 3.2), including CDERs. Furthermore, when a user job executes, it does not expect to contend for the same CPU resource with other user jobs – this exemplifies *Exclusivity*. Exclusivity is a fundamental feature of both Operating System resource management and database management, it helps protect against unintended side-effects when one or more users attempt to access the same resource concurrently.

It is evident that when integrating new CDERs into the Grid it would be desirable to preserve the Discovery, Independence and Exclusivity properties. Using these properties as a basis for addressing the resource integration challenge, the following *Grid Resource Integration Principles* are proposed:

- **Discovery:** The resource should be published as one or more GLUE entities. The act of publishing the resource into the Grid Information System makes the resource discoverable by users and services. An entity should, where possible, contain attributes that reflect the resource's properties, and also where possible, these attributes should be *quantitative*. This allows resources of the same type to be compared and ranked against each other. (For example, the vendor, model and performance characteristics of the resource may be of importance when selecting appropriate resources for a job.)

- **Independence:** There should be a method through which the required resource can be specified using a job description language. The specification should be

independent of the way in which the resources are locally managed. (For example, the mechanism used to specify GPGPU requirements to the TORQUE/MAUI scheduling system differs from that used by SLURM.)

- **Exclusivity:** A resource allocated to a job by a batch system should be available as if it were exclusively allocated to the job. (For example, a GPGPU allocated to a job should not be available or even visible to another job running on the same worker node.)

Resources that satisfy these conditions will be regarded as *first-class* because their usage on a Grid is similar to the CPUs. Despite the elevated role of the CDER in the execution of its workload, it should be noted that one or more CPUs are still required.

CDER Accounting and Monitoring

Two further properties should be regarded as important in the handling of CDERs. These are called *Accounting* and *Monitoring*:

- **Accounting:** Where possible, it is desirable to have the facility to record detailed information about the usage of the CDER, such as the amount of time the CDER dedicated to the job's processing (system time), how much memory was used, and what user executed the job.

- **Monitoring:** Where possible, there should be mechanisms to query the internal state of the CDER for failures. Further monitoring may be desirable (e.g. testing the correctness of software that is needed to access the CDER).

The role of resource usage accounting is important in Grid infrastructures: firstly, it keeps track of who has used the resource, so there is an audit trail in the case of any problems (e.g. security violations or application debugging); secondly, the LRMS keeps track of how long a user's job has utilised its allocated resources. The resource providers can extract this information and calculate which users or groups of users have been using their resources, and for how long. Grid services such as APEL [137](EGI) and Gratia [138](OSG) provide a way for resource providers to analyse and publish the CPU resource usage of each Virtual Organisation. This also allows Virtual Organisations to ensure that any service level agreements have been met. Furthermore some Grids, notably EGI, are investigating a pay-per-usage charging model [139], so

resource accounting will have a more critical role in existing grid infrastructures. Detailed CDER accounting may not always be possible. For example, low- and mid-range GPGPUs (e.g Nvidia GTS 450) do not support per process accounting, so it is difficult to gauge how much time a task or job spent running on the GPGPU itself. Indeed, even the high-end GPGPU virtual machine offered through the Amazon Cloud service is charged according to the type of the particular virtual machine used, and for how long it was used – Amazon does not charge based on actual GPGPU usage. Detailed CDER *Accounting*, whilst desirable, is not always possible.

Monitoring seeks to determine the health of resources and services by running selected tests on a regular basis (e.g every minute or hour). Grid monitoring should also alert the resource provider when a failure has occurred. This allows the resource provider administrators to react quicker and therefore helps minimise service disruption. Monitoring can also help users that rely on direct job submission with a means of selecting good resource providers (by observing/using the monitoring data). Furthermore, monitoring systems such as Nagios [140] provide visual feedback and maintain historical availability and reliability data (which can be used to validate service level agreements).

Accounting and Monitoring are integral components in the operation and management of Grid resources. However their study is outside the scope of this thesis, primarily because:

(i) the thesis is concerned with CDER resource management; and

(ii) neither Accounting nor Monitoring are involved in CDER job execution.

The CDER definition assumes exclusive allocation of the resource (p.73). In this case, it can be implied that if a single CDER is allocated, then it is allocated for the duration of the job. The resource usage can be accounted for by using the CPU accounting data. However, this scheme will only work for single-CPU/single-CDER jobs, and does not work for other cases. The basis of a potential solution to this problem is proposed in Appendix B. The solution exploits existing operating system resource auditing capabilities. Such auditing can record detailed information about access to selected hardware. In this way it can be used to determine which user has accessed the hardware, and for how long.

The Monitoring property can be fulfilled by developing a test suite of small, but simple, CDER-based applications. This test suite should examine common CDER-

specific job requirements.

The OGF GLUE Schemas: Conceptual and Concrete models

As noted above, the GLUE Schema and the Grid Information System (p. 51) are key components in facilitating the discovery of resources on the Grid.

The developers of GLUE 1.3 note [103] that the full set of features and policies for a given LRMS is much too complex to be represented in a reasonably compact schema. Furthermore, because LRMS implementations have features that vary qualitatively, the schema definition is intended to capture the most-common configurations among the supported LRMS's. The same concerns and considerations also apply to GLUE 2.0. An unintended consequence of this approach is that neither GLUE 1.3 nor GLUE 2.0 provide native support for CDERs, i.e. there is no CDER (or equivalent) GLUE resource type. This is because (i) CDERs (without resource discovery support) are usually implemented as a generic consumable resource in the LRMS (p. 16), and this was not supported in LRMS scheduling systems such as MAUI; and, (ii) Grids developed around CPU-based processing, so the use of CDERs for massively-parallel processing was not foreseen,

Extending GLUE Entities

Both the GLUE 1.3 and GLUE 2.0 schemas provide ways to associate additional data with existing GLUE entities. The main differences are (i) GLUE 1.3 Attribute-Extensions are limited to a few GLUE classes through the *capacity* attribute, whereas under GLUE 2.0, all Classes (except Extensions) can be extended by adding one or more *OtherInfo* attributes. (ii) GLUE 1.3 allows for Services to be extended by using instances of a GlueServiceData Entity; under the LDAP rendering, the Key and Value can be associated with a Service Entity instance by using a *GlueChunkKey* – this extends Service instances only, so there are clear limitations in how this can be used. In contrast, the GLUE 2.0 LDAP rendering allows all object instances to be extended using one or more Extension instances. In this regard, GLUE 2.0 has a clear advantage over GLUE 1.3 when publishing CDER-related information.

3.2.1 Conceptual CDER Discovery Strategies

Several conceptual strategies have been identified that would allow grid jobs to discover and select CDERs on grid infrastructures using existing GLUE schema. Each of the strategies presented describes (i) an approach for publishing information describing a CDER and (ii) a method for using this published information to satisfy the specific CDER requirements of a grid job.

A-Priori Strategy

This strategy does not publish any CDER GLUE data and requires that the grid user has prior knowledge of the specific CDER, its properties, the name of queue used to access it, and any additional access requirements. This is effectively a 'null-strategy' and is the only-available CDER access method in the absence any other strategy. It may, however, be a useful approach, for example, when testing the deployment of new CDERs.

When submitting a job using this strategy, the user must specify exactly the grid queue where the job is to be executed. For example, this method has been used to support the execution of GPGPU applications at selected Sites on the EGI [72].

Named-Queue Strategy

A community-adopted queue-naming convention may be used to indicate that a job requirement is satisfied or that a resource is attainable by using that queue. For example, the queue name suffix `sdj` (Short Deadline Job) has been used to advertise grid queues that support high-priority job execution [141]. Similarly, the suffix `gpgpu` could be used to imply that GPGPU CDERs are available through a specific queue. Listing 3.1 demonstrates the specification of a requirement that a job be submitted to a queue with the `gpgpu` suffix using the gLite/UMD middleware job description language (JDL).

Tagged-Environment

More-versatile strategies are possible if the information schema allows arbitrary new information to be published alongside the information describing the existing resources. For example, it is common in grid information schemas that environment tags may be used to advertise that specific software is supported at a Resource Centre. They

Listing 3.1: An example Named-Queue grid job specification

```
[
Type="Job";
JobType="Normal";
Executable = "myScript.sh"; # Script to invoke GPGPU application
StdOutput = "std.out";
StdError = "std.err";
InputSandbox =  {"myScript.sh"};
# Regular expression to match all queue names ending with 'gpgpu'
Requirements = ( RegExp(".*gpgpu$", other.GlueCEUniqueID) );
]
```

may also be used to advertise hardware configurations. This strategy is an implicit mechanism for adding new attribute values, and it may also be used to publish arbitrary information about the CDER. The Tagged-Environment strategy is widely used on the EGI to support the execution of multi-core applications using MPI-START [8] (e.g., GLUE 1.3 SoftwareEnvironment tags have been used to advertise the availability of Infiniband networking [142]).

A concrete example showing how GPGPU CDERs might be accessed is as follows: if a Resource Center publishes the tags shown in Listing 3.2, then a user requiring Nvidia Kepler GPGPUs can specifically target those Resource Centers by using the *Requirements* expression in Listing 3.3.

Listing 3.2: An example GLUE 1.X Tagged-Environment advertising both software (CUDA) and hardware (NVIDIA-KEPLER) capabilities.

```
GlueHostApplicationSoftwareRunTimeEnvironment: CUDA
GlueHostApplicationSoftwareRunTimeEnvironment: NVIDIA-KEPLER
```

Listing 3.3: Example Tagged-Environment job specification requiring the NVIDIA-KEPLER hardware capability

```
[
...
Requirements = (Member("NVIDIA-KEPLER",other.
   ↪ GlueHostApplicationSoftwareRunTimeEnvironment));}
]
```

Attribute-Extension

Using this strategy, CDERs are explicitly associated with existing grid resources (e.g., worker nodes), whose properties are already captured by the information schema (e.g.,

ExecutionEnvironment instances). The schema may allow the description of an existing resource to be *internally* extended with new attributes and their associated values. Under GLUE 2.0, Attribute-Extension can be implemented using OtherInfo attributes. An arbitrary number of OtherInfo attributes may be applied to any GLUE 2.0 Entity. Listing 3.4 provides a concrete example of the use of this strategy to publish detailed information about a GPGPU CDER. The GPGPU is explicitly associated with an ExecutionEnvironment instance. The information published is both static – describing hardware characteristics – and dynamic – reflecting current capacity and utilisation. Each key and value pair are encoded into a single string. This has implications for querying CDER states.

Listing 3.4: An example of the publication of static and dynamic GPGPU information by extending GLUE 2.0 an ExecutionEnvironment instance using OtherInfo attributes.

```
objectClass: GLUE2ExecutionEnvironment
...
GLUE2EntityOtherInfo: GPGPUTotalInstances=32
GLUE2EntityOtherInfo: GPGPUUsedInstances=2
GLUE2EntityOtherInfo: GPGPUCUDAComputeCapability=2.1
GLUE2EntityOtherInfo: GPGPUMainMemorySize=1024
GLUE2EntityOtherInfo: GPGPUMP=4
GLUE2EntityOtherInfo: GPGPUCoresPerMP=48
GLUE2EntityOtherInfo: GPGPUCores=192
GLUE2EntityOtherInfo: GPGPUClockSpeed=1660
GLUE2EntityOtherInfo: GPGPUECCSupport=false
GLUE2EntityOtherInfo: GPGPUVendor=Nvidia
GLUE2EntityOtherInfo: GPGPUPerNode=2
```

The Attribute-Extension strategy has been previously used in a prototype UMD-based testbed to publish GPGPU information in GLUE 2.0 ApplicationEnvironment instances [14]. Current implementations of job orchestration systems (e.g., the UMD WMS) will be unable to process CDER extensions to satisfy job requirements, so an alternative two-phase job submission mechanism is proposed. This will be described in Section 3.2.2.

Class-Extension

A by-reference alternative to the by-value Tagged-Environment and Attribute-Extension strategies may be used if the information schema allows the information describing existing resources to *externally* reference information describing CDERs. For example, any GLUE 2.0 Entity may be associated with zero, one, or more instances of the Extension class, each of which contains a single Key/Value pair. These Key/Value pairs

may be used to describe the properties of an associated CDER. This is illustrated in Listing 3.5. Both the key and value are managed as separate attributes, so it is significantly easier to query and parse CDER state data under the Class-Extension strategy than under the Attribute-Extension strategy. Like the Attribute-Extension strategy, this strategy requires the two-phase job-submission mechanism proposed in Section 3.2.2.

Listing 3.5: A single GLUE 2.0 Extension instance that is associated with a parent ComputingShare instance. The extension is used to publish the `GPGPUPerNode=2` Key/-Value pair.

```
dn: GLUE2ExtensionLocalID=GPU_NVIDIA_P_1,GLUE2ShareID=
   ↪ gpgpu_gputestvo_ce.example.com_ComputingElement,GLUE2ServiceID=
   ↪ ce.example.com_ComputingElement,GLUE2GroupID=resource,o=glue
GLUE2ExtensionLocalID: GPU_NVIDIA_P_1
objectClass: GLUE2Extension
GLUE2ExtensionKey: GPGPUPerNode
GLUE2ExtensionValue: 2
GLUE2ExtensionEntityForeignKey: gpgpu_gputestvo_ce.example.
   ↪ com_ComputingElement
```

3.2.2 Two-Phase Job Submission

The CDER discovery strategies presented above are intended to be used to help grid users or services identify where the CDERs are deployed, as well as their capabilities, usage, and state. By using the published data, grid jobs can be restricted to matching Sites. However, current implementations of job orchestration systems, such as the Workload Management System (p. 49), cannot exploit the published CDER data.

Rather than having to update workload management systems to handle every new type of CDERs, an alternative, but flexible, two-phase approach is proposed. The first phase searches the Grid Information System to identify resource providers that match the user's CDER requirements, and the second phase submits the job to the Grid as normal. Figure 3.3 illustrates a high-level overview of the two-phase process, allowing grid users to submit jobs with CDER requirements. Both phases are initiated from the *User Interface* – the machine from which grid jobs are submitted.

Phase-1:

1. The CDER requirements are extracted from a CDER Job Description file and are used to determine the type of CDER required (Figure 3.3, Step 1).

2. A suitable GIS query is constructed based on the CDER type, and is then sent to the GIS (Figure 3.3, Step 2). This step is independent of any job-orchestration system, such as the WMS.

3. The query results are returned to the User Interface for further processsing (Figure 3.3, Step 3).

4. User-specified CDER requirements are matched against the retrieved CDER descriptions as follows:

   (a) The individual records of the GIS response are parsed. The parsing is used to extract the CDER attribute key/value pairs and the name of the Compute Element managing the CDER. The extracted data is used to construct a *Resource Offer*[1].

   (b) Similarly a *Resource Request* is created from the job's CDER requirements.

   (c) The Resource Request is matched against each of the Resource Offers to determine if the Compute Element satisfies the original CDER requirements.

   In this way, a list of all Compute Elements satisfying CDER requirements can be constructed. This list of matching Compute Elements is used as the basis of a refined and targeted job requirement expression (Figure 3.3, Step 4).

Phase-2:

5. A modified job description file is submitted to a workload management system in the usual manner. This modification includes the targeted Compute Element requirement expression, and any CDER job requirements that need to be passed into the selected Compute Elements LRMS (e.g. The number of required CDER resources). The targeted Compute Element requirement expression has the effect of restricting the workload management system to considering only those Compute Elements that matched the CDER requirements in Phase-1.

[1]Resource Offer and Resource Request are both terminology used by ClassAd [143]. ClassAd is a match-making software used in several Grid brokering systems such as HTCondor and gLite.

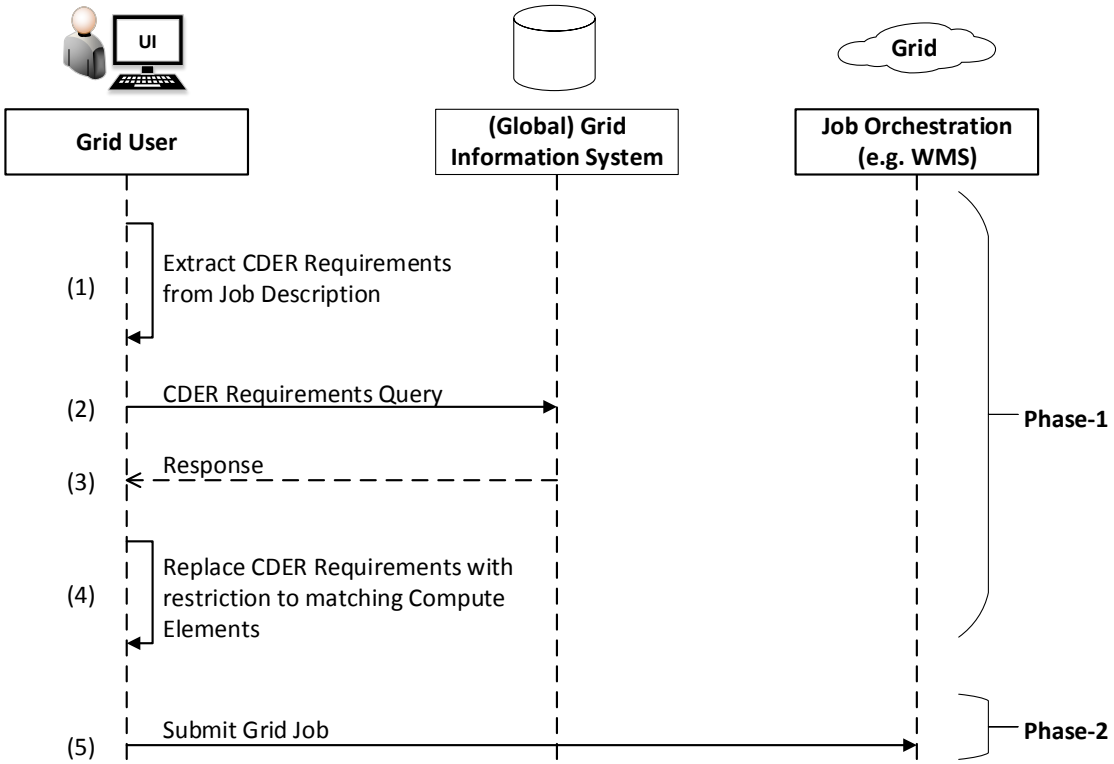


Figure 3.3: An abstract model of the two-phase submission of a grid job with CDER requirements.

To summarise, Phase-1 is to refine the job requirements to a targeted expression, and Phase-2 is to submit the targeted job to a job-orchestration system (such as the WMS). It should be noted that Phase-1 does not involve job orchestration.

3.2.3 Analysis

The conceptual strategies presented in Section 3.2.1 introduced five different approaches that may be taken when integrating CDERs into grid environments. Furthermore, some concrete examples have been presented to illustrate how these strategies may be used in practice. This section presents a methodology for comparing these strategies. The methodology sets out several criteria which are used to classify particular strategy properties. An analysis of the strategies is summarised in Table 3.1 and discussed in detail later. To aid the comparison, each of the strategies is applied to a representative test case (a typical GPGPU resource) using an LDAP rendering. This test case is used to compare the cost of each strategy with respect to data size and query time.

Table 3.1: Summary of CDER Strategy Evaluation

	A-Priori	**Named-Queue**	**Tagged-Environment**	**Attribute-Extension**	**Class-Extension**
Discovery	No	Yes	Yes	Yes	Yes
Semantic Resource Detail	None	Minimal	Coarse	Fine	Fine
Semantic Structure	None	Minimal	Minimal	Yes	Yes
Dynamic Information	No	Minimal	Deprecated	Yes	Yes
GLUE 1/2	Any	Any	1.3	2.0	2.0
Time Efficiency	N/A	N/A	Low	Med	High
Space Efficiency	N/A	N/A	N/A	High	Low
Two-Phase	No	No	No	Yes	Yes

Methodology

Each of the conceptual strategies proposed in Section 3.2.1 is evaluated below, with respect to the following criteria:

Discovery It is only possible to discover previously unknown CDERs if the strategy used publishes at least some minimal information to identify the resource in the information system.

Semantic Resource Detail The ability of a strategy to publish semantic detail beyond the mere existence of a particular type of CDER would enable more-powerful job requirement satisfaction. As an example, a CDER such as a GPGPU may have several intrinsic properties (e.g. GPGPU memory size and model) that are interesting for resource selection, so the strategy must be capable of publishing such details.

Semantic Structure Similarly, the ability of a strategy to associate CDERs with other grid entities will again facilitate more-powerful resource selection.

Dynamic Information While certain CDER characteristics (e.g., those describing physical hardware properties) will remain static, dynamically changing properties such as current availability and utilisation will also be important considerations

in resource selection. Strategies will vary in their ability to dynamically update such information.

GLUE Version Each strategy may be more or less appropriate to each version of the GLUE schema.

Information Time Efficiency The effort required to (i) satisfy the CDER requirements for a specific grid job, and (ii) construct a complete representation of the current state of a CDER is considered for each strategy. If no data is published, then this is deemed *Not Applicable (N/A)*.

Information Space Efficiency Conceptually, the properties of a CDER are published as Key/Value pairs. The overhead of publishing this data is considered under this heading.

Matchmaking/Job Submission Support Some strategies will allow CDER requirements to be satisfied directly using existing job specification and submission mechanisms, while other strategies will require an extended mechanism, such as the two-phase job submission approach described in Section 3.2.2.

In order to illustrate and quantitatively compare the strategies, the evaluation references a contemporary example of a GPGPU CDER. A schema representing a set of typical GPGPU properties and (GPGPU-related) LRMS properties is tabulated in Table 3.2. The Sample Values describe the properties of 32 identical Nvidia GTS 450 GPGPUs installed on 16 worker nodes with 2 GPGPUs per worker node.

Experiment 1: Data Publication Cost

The first quantitative experiment examines the total data size of the published CDER information when using each of the the strategies under an EGI BDII/LDAP-based realisation. The summary results are presented in Table 3.3.

Experiment 2: Timed CDER Matchmaking

A second quantitative experiment measures the time taken to determine the set of ComputeElements (i.e LRMS queues on the Grid) that satisfy a typical CDER property. Only GLUE 2.0 strategies that satisfy the *Fine Semantic Resource Detail*, *Semantic*

Table 3.2: Schema representing typical properties of a GPGPU CDER

Sample GPGPU attributes	Type	Source	Creation	Sample Values
GPGPUTotalInstances	Integer	LRMS	Dynamic	32
GPGPUFreeInstances	Integer	LRMS	Dynamic	30
GPGPUPerNode	Integer	LRMS	Static	2
GPGPUCUDAComputeCapability	Float	GPGPU	Static	2.1
GPGPUMainMemorySize	Integer	GPGPU	Static	1024
GPGPUMP	Integer	GPGPU	Static	4
GPGPUCoresPerMP	Integer	GPGPU	Static	48
GPGPUCores	Integer	GPGPU	Static	192
GPGPUClockSpeed	Integer	GPGPU	Static	1660
GPGPUECCSupport	Boolean	GPGPU	Static	false
GPGPUVendor	String	GPGPU	Static	Nvidia
GPGPUModel	String	GPGPU	Static	GTS_450

Table 3.3: Overhead of publishing typical data under LDAP Data Interchange Format

Strategy	Byte-count
A Priori	N/A
Named-Queue	N/A
Tagged-Extension (GLUE 1.3)	800
Tagged-Extension (GLUE 2.0)	10500
Attribute-Extension	800
Class-Extension	4800

Structure, and *Dynamic Information* criteria are considered. The experiment methodology is as follows:

(i) Generate a snapshot of the GLUE 2.0 data from an EGI Top-Level BDII;

(ii) The sample data from Table 3.2 is used to add GLUE 2.0 data representing CDERs (that don't presently exist) in the form required for that strategy. In the case of the Attribute-Extension strategy, OtherInfo attribute data is inserted into a sample set of objects. In the Class-Extension strategy case, a set of Extension objects are generated, consisting of Key, Value and Foreign Key; and

(iii) The amended GLUE 2.0 is copied to a modified testbed Top-Level BDII as a snapshot. This modification to the BDII disables any further updates to the GLUE data, ensuring the persistence of the experimental data in the Grid Information System.

The data presented in Table 3.4 is the average system-time cost (measured in seconds) of matching a sample ClassAd GPGPU Resource Request against 100 Resource

Offers generated from GLUE 2.0 BDII LDAP queries. The Schema in Table 3.2 was used to generate the GLUE 2.0 data for Attribute-Extension and Class-Extension Entities. This data was appended to 100 randomly selected GLUE2ComputingShare Entities (Attribute-Extension strategy). Similarly, 1200 Glue2Extension objects were generated to extend 100 GLUE2ComputingShares (Class-Extension strategy). The time includes the cost of retrieval, ClassAd [143] Resource Offer generation, matchmaking, and returning a list of matching Compute Elements.

Table 3.4: Average system-time cost, measured in seconds, of matching a sample ClassAd GPGPU Resource Request against 100 Resource Offers generated from GLUE 2.0 BDII LDAP queries.

Strategy	Time (sec)
Attribute-Extension	1.26
Class-Extension	1.31

The baseline EGI GLUE 2.0 data used to populate the Top-Level BDII is representative of the state of a very large grid infrastructure. Indeed, an EGI BDII currently contains more than 10^5 GLUE 2.0 Entities. The modified BDII's snapshot of this data is intended to simulate querying and discovering CDERs in Grids with a similar number of GLUE objects to EGI.

A-Priori Strategy

The A-Priori strategy is clearly contrary to the *Discovery* Grid Resource Integration Principle (p. 76), because no data relating to the CDER or the LRMS is captured or published. This method must be deemed unsuitable for use for handling CDERs except for test purposes.

Named-Queue Strategy

The Named-Queue strategy requires that, at a minimum, the name of the queue is used to encode the nature of the CDER. This strategy is one of the simplest methods available to publish/discover particular CDER types. However, it does not work well as a method to encode the other properties listed in Table 3.2. Support for *Semantic Resource Detail* and *Semantic Structure* is *Minimal*. Furthermore, *Dynamic Information* cannot be encoded into the queue name. Consequently, this method allows very limited resource discovery.

One method that may be used to publish the capacity and utilisation of CDERs is to configure the LRMS with a 1:1 binding between a CPU core and a dependent CDER. Unbound CPUs must then be configured to be unavailable to the LRMS. The effect of applying this configuration is that the capacity and utilisation of CDERs can be directly determined from the capacity and utilisation of the CPUs. A negative consequence of this configuration is that it may result in CPU under-utilisation on many-core systems. This is due to the unavailability of the unbound and non-allocatable CPUs. This configuration also results in a condition whereby certain job requirements cannot be met despite the availability of sufficient resources (again, due to the unbound CPUs).

Tagged-Environment Strategy

Neither of the first two strategies support the publication of arbitrary information describing CDERs. In contrast, the remaining Tagged-Environment, Attribute-Extension, and Class-Extension strategies enable the publication of information with greater semantic detail. The first of these – Tagged-Environment – allows Sites to publish arbitrary data enabling basic CDER discovery. The limitations of this strategy are: (a) there are no implicit relationships between published tags, and any relationship between tags must be reconstructed by other means; (b) dynamic data is possible but impractical as the tags are normally treated as static values, which are used to advertise the presence of particular software on worker nodes. Encoding frequently changing CDER capacity or usage data into a tag implies removing old tags and adding new ones. Hence, this method cannot be recommended.

If the schema in Table 3.2 were to be published as GLUE 1.3 SoftwareRuntimeEnvironment attributes, then the production of the sample GLUE data generates approximately 800 bytes – a relatively small amount of data in comparison to the Class-Extension strategy (4800 bytes).

Listing 3.6: Sample Data for Tagged-Environment Strategy

```
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUTotalInstances=32
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUFreeInstances=30
GlueHostApplicationSoftwareRunTimeEnvironment: GPUPerNode=2
GlueHostApplicationSoftwareRunTimeEnvironment:
   ↪ GPUCUDAComputeCapability=2.1
GlueHostApplicationSoftwareRunTimeEnvironment: GPUMainMemorySize=1024
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUMP=4
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUCoresPerMP=48
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUCores=192
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUClockSpeed=1160
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUECCSupport=false
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUVendor=Nvidia
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUModel=GTS_450
```

There are, however, some negative effects that have been observed when evaluating this approach on the EGI. Namely, GLUE 1.3 SoftwareRuntimeEnvironment attributes are automatically converted by the UMD grid-middleware into individual GLUE 2.0 ApplicationEnvironment object instances. This GLUE format conversion will continue until the Grid infrastructure fully migrates from GLUE 1.3 to GLUE 2.0. Using the the sample values from Listing 3.6, the GLUE2 information provider produced over 10.5 KB for the 12 ApplicationEnvironment objects – over 800 bytes per object. Note that, as the SoftwareRuntimeEnvironment tags do not exist in native GLUE 2.0, this strategy is only applicable to GLUE 1.3.

Attribute-Extension Strategy

This strategy uses GLUE 2.0 OtherInfo attributes to insert an arbitrary number of Key/Value pairs into selected Entity instances. In this way, objects can be rich in *Fine Semantic Resource Detail*. Furthermore, these Key/Value pairs can be applied to different entities as appropriate – CDER hardware properties can be applied to ExecutionEnvironment instances, and capacity and utilisation data can be applied to VOShare instances, so *Semantic Structure* is supported. The individual attribute values in object instances can be *Dynamically* updated, but only by converting the Key/Value pair to a string and replacing the old string with the new one. This has an impact on the *Time Efficiency*. All GLUE 2.0 Entities can be extended in this strategy by using OtherInfo, but support under GLUE 1.3 is limited to a few Entities

that support *capability* attributes. *Space Efficiency* is *High* as data publishing costs are similar to the GLUE 1.3 Tagged-Environment strategy (approximately 800 bytes), but would increase linearly (by less than 160 bytes per VOShare). To use the CDER data to submit grid jobs, the two-phase method must be used.

Class-Extension Strategy

An arbitrary number of instances of the Extension class can be created in GLUE 2.0. Each consists of a key, a value, and a *Foreign Key* reference to the object instance that it extends. As a result, the basic *Discovery*, *Fine-grained Semantic Resource Details*, and *Semantic Structure* criteria are satisfied. *Dynamic Information* is supported by updating the Extension value. Furthermore, discovering Keys or Values are low-cost operations, so the *Time Efficiency* is *High*. There is an overhead in the creation of each Extension instance, as each Extension (and its Key/Value/Foreign Key) is encapsulated with extra LDIF data. This encapsulation data is generally several orders of magnitude greater than both the key and value. For the sample data, the average number of bytes for both the key and values was approximately 17 bytes, but encapsulating a single Key/Value pair in an object costs in the order of 400 bytes (including the Foreign Key). So, *Space Efficiency* is *Low* in comparison to Attribute-Extension, which is counter-intuitive for a by-reference mechanism. To use the CDER data to submit grid jobs, the two-phase method must be used.

Analysis Conclusions

It is very clear that the *A Priori*, *Named-Queue* and *Tagged-Environment* strategies have very limited use in modern Grids that want to use CDERs. Some of these strategies can be used to aid the discovery of whether a resource exists, but they do not have a flexible way to convey rich detailed information (describing many values or properties) nor handle dynamic capability and capacity information. The remaining *Attribute-Extension* and *Class-Extension* strategies both allow CDERs to publish rich static and dynamic information. The Attribute-Extension is best for publishing blocks of related static data (significantly less data is published in comparison to the Class-Extension strategy), while Class-Extension is better for publishing small sets of dynamic data that can be queried without the need for additional parsing. Both strategies can be used as part of an alternative (and flexible) two-phase process that supports CDER

discovery and CDER-based job submission without the need for major changes to the grid middleware.

3.3 CDER Case Study: GPGPU Integration

Having defined the concept and properties of a CDER; examined what strategies are available to publish CDER details on the Grid; and having proposed a two-phase job submission workflow method that allows the CDER to be discovered, matched and allocated (i.e. handle job orchestration using CDER resource job requirements), this section looks at how to apply this conceptual approach to a sample CDER on a Grid.

It was already known from the EGI GPGPU User and Site Administrator Surveys (Section 2.4.3) that GPGPUs would be one of the most important types of CDER that users and resources providers would expect to have access to on the Grid. This means that the reason for selecting GPGPUs as a real-world implementation target is well-founded. Furthermore, the potential number of grid middleware and LRMS combinations is large (e.g. Globus and HTCondor, ARC and SLURM), but there are finite time limits that restrict how many of these combinations can be investigated in this thesis. The study of the middlware/LRMS deployment of a contemporary Grid (EGI) in Section 2.4.4 showed that the CREAM Compute Element was deployed on 427 of 567 instances (or 75%). TORQUE is by far the most popular choice of LRMS, and is deployed on 316 of 567 Compute Elements (55.7%). The number of Compute Elements that employ a TORQUE-based CREAM combination amounts to almost 51% of all Compute Element instances (see also Table 2.7). This indicates that the most compelling Compute Element to develop a GPGPU CDER solution for is the TORQUE/CREAM Compute Element.

Before attempting to apply the *Grid Resource Integration Principles* in full to GPGPU CDERs, an examination of the basic user and system requirements is carried out:

Discovery: The salient properties that help describe a GPGPU are similar to those used to describe CPUs: vendor, model, memory, speed, benchmarked performance, number of physical GPGPUs per worker node. LRMS properties that can influence WMS orchestration and ranking include the total number of installed GPGPUs and the number that are currently allocated through the LRMS. Users may also be interested

in the software required to access the resource and basic installation details. These properties could be advertised in the Grid Information System as one or more GLUE entities and optionally used as a filter during resource selection.

Independence: The user needs an LRMS-independent way to specify the number of GPGPUs required or the number of GPGPUs that the job needs per-CPU core (this implies a minimum number of GPGPUs per worker node).

Exclusivity: The user needs assurances that two or more jobs concurrently executing on the same worker node cannot use the same GPGPU. In the absence of batch-system support for such exclusivity, an additional mechanism must be provided.

Prototype Implementation

As shown in section 2.4.1, the mechanism for orchestrating the execution of a grid job follows a complex chain of events. As a result, providing access to new grid resources, such as GPGPUs, is non-trivial. In particular, if these resources are to be provided by *existing* grid infrastructures that are *in-production* and in continuous use by an extensive community of users, the challenge becomes more acute. Adding support for new resources cannot be dependent, for example, on architectural changes, the replacement of core services or modifications to the GLUE schema. Instead, the approach taken must integrate with existing infrastructure, while adhering to the principles of *discovery*, *independence* and *exclusivity* (Section 3.2).

To minimise the number of changes to the core grid middleware, the solution proposed in this prototype is therefore to provide a set of modular *hooks*. The hooks provide a simple and extensible method that (i) allows newer CDER resource types to be added; and (ii) allows external applications to produce GLUE-compliant data. In this section, it will be shown that this approach requires relatively small changes to existing grid middleware to produce the CDER GLUE data.

Prototype Infrastructure

The focus of this prototype is on the integration of Nvidia GPGPUs using the CUDA runtime and application development framework. (Nvidia GPGPUs are the most widely used GPGPU in High Performance Computing centres.) The prototype was

developed using the UMD gLite grid middleware, and consists of a User Interface (UI), a Workload Management System (WMS), a Top-Level BDII (BDII), grid-security infrastructure services, and a Resource Centre using the CREAM CE. The CREAM CE uses TORQUE 2.5.7 as the LRMS together with the MAUI resource scheduler.

Consumable Resources with MAUI

One of the more immediate problems with TORQUE is that it is normally combined with the MAUI resource scheduler by cluster or grid resource providers [144]. MAUI supports many advanced features, such as CPU-based *fair share* allocations. Fair share support allows jobs to be scheduled or prioritised according to a defined policy that takes (time limited) historical job allocation information into account. However, as noted in Table 2.1 TORQUE/MAUI does not correctly handle GPGPU allocation, nor indeed, does it have good support for defining new resource types. Worse still is that the MAUI open-source product has been deprecated by its maintainers (Adaptive Computing) in favour of the commercial product MOAB [26]. Licence restriction on how MAUI can be modified and distributed make it difficult for the LRMS administrators or grid community to provide additional community updates [144].

To overcome the problem of MAUI handling of GPGPUs and other consumable resources, the prototype was modified with a publicly available source-code patch [145] to enable consumable resources (i.e. user defined resources).

3.3.1 Discovery: GPGPU Schema and Information Providers

In the conclusion of the evaluation of the conceptual strategies (p. 92), it was noted that the *a priori* strategy has no facility for resource discovery, while *Named-Queue* and *Tagged-Environment* strategies have very limited use – they can be used to determine if a resource exists, but they do not have a flexible way to convey rich detailed information (describing many values or properties), nor can they handle dynamic capability and capacity information. In addition, these methods lack inherent middleware support that allow users or grid services to discover detailed information about GPGPU resources, or to submit jobs that are capable of using highly-tailored GPGPU resource requirements (e.g. based on GPGPU benchmarks or memory).

Conveying the rich set of GPGPU information described in Table 3.2 using the GLUE schema therefore requires the use of either the Attribute-Extension strategy,

the Class-Extension strategy or a combination of both. In this prototype, a simple representation of these details is realised by using the GLUE 2.0 *ExecutionEnvironment* entity [104]. This entity is primarily used to describe the properties of cluster resources managed by an LRMS, such as the type of CPU used and its SPECint and SPECfp benchmark performance. This makes it a reasonable choice of entity to extend with the GPGPU CDER information using an Attribute-Extension strategy.

In this prototype the existing ExecutionEnvironment Generic Information Provider (GIP) is replaced with a new Information Provider that facilitates per-CDER *hooks*. For each CDER, separate hooks can be provided to generate static and dynamically changing GLUE data. This is illustrated in Figure 3.4. In particular, the prototype modifies the ExecutionEnvironment GIP by adding a new hook to publish static data – this pertains to static GPGPU hardware properties. The hook also supports gathering and publishing previously unavailable GPGPU capacity and utilisation data from MAUI dynamically. Listing 3.4 (p. 82) illustrates an example of some of the output generated by the execution of a modified Information Provider where both dynamic and static hooks for the LRMS ExecutionEnvironment have been added.

The approach taken here differs slightly from the work presented in the paper "Supporting job-level secure access to GPGPU resources on existing grid infrastructures" [14]. The primary difference is the change of GLUE entities to represent the CDERs. The original work used an ApplicationEnvironment entity because it defined FreeJobs and MaxJobs fields. These are non-mandatory values that were respectively used to publish the number of Free and Total GPGPU CDERs available. Both CUDA and OpenCL ApplicationEnvironments were defined. This gave rise to a number of weaknesses: (i) there was an unnecessary duplication of CDER data; and (ii) using a GLUE entity (ApplicationEnvironment) that describes software to publish hardware details was incorrect. It did, however, greatly simplifly the publication of arbitrary data in the orginal prototype.

3.3.2 Independence: Satisfying GPGPU Job Requirements

The Independence principle requires that the grid user should be able to specify GPGPU resource requirements within the existing job submission framework in a manner that is independent of any Compute Element LRMS implementation. If the requirement that a user can request a given number of GPGPUs per worker node is

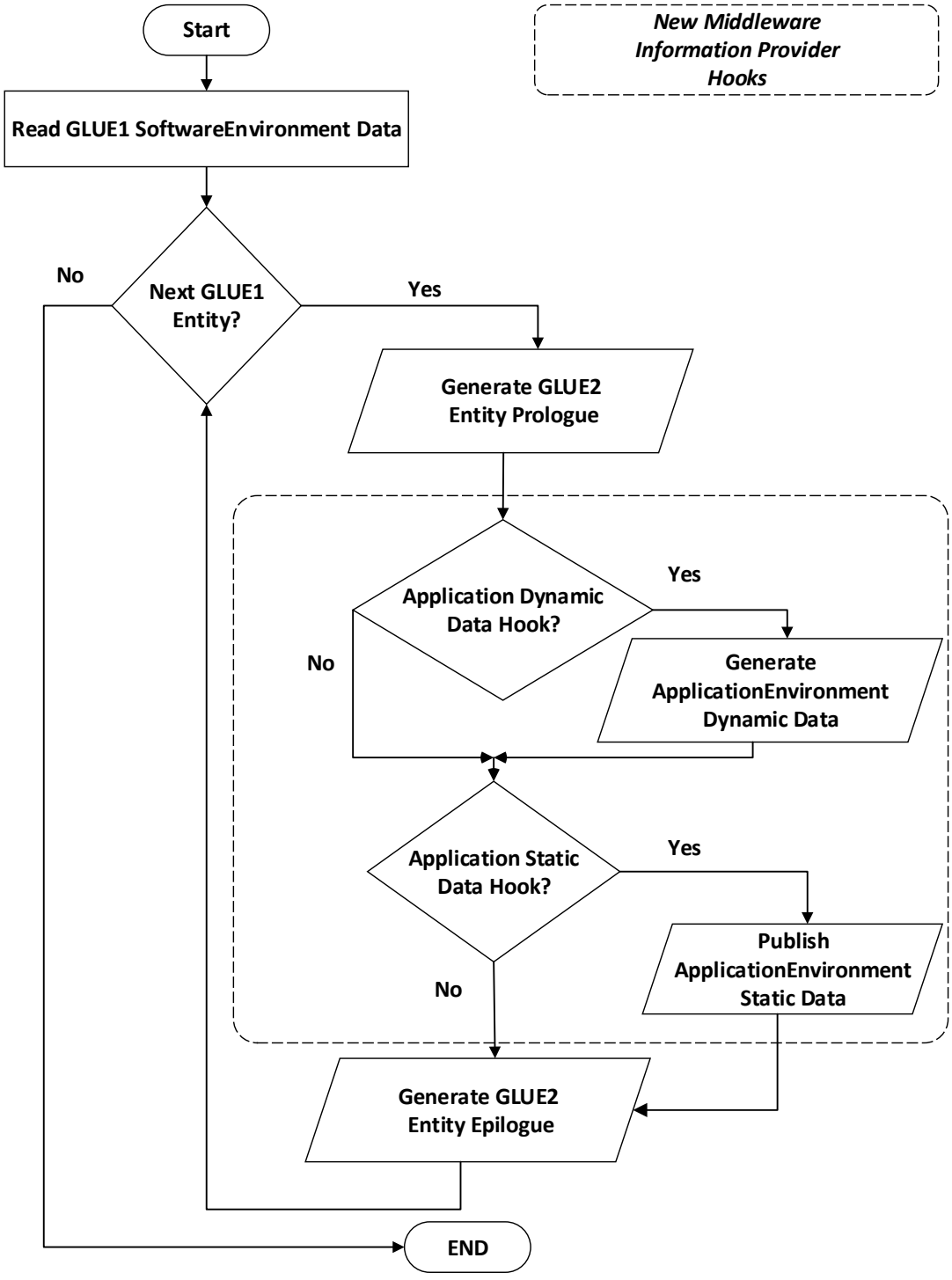


Figure 3.4: Workflow of Modified ExecutionEnvironment Information Provider

taken as an example (a requirement gathered from the GPGPU Surveys), it should be possible to request it with a JDL expression similar to:

```
GPGPUPerNode=X; # Where X is a positive integer
```

Independence then requires that this expression be converted into a resource request suitable for any choosen Compute Elements LRMS. In the modified TORQUE/MAUI case, this expression could take the form of:

```
#PBS -W 'x=GRES:gpus@X'
```

The gLite middleware has an existing provision that allows a limited amount of GLUE-related data to be passed into a given LRMS. This process is known as *Forwarding Requirements to the Batch System* [146]. For example, it can be used to pass additional requests into an LRMS that specify that the LRMS must select worker nodes that satisfy a given property. Despite its usefulness, the method has some significant weaknesses:

1. Each WMS must modify a configuration file to allow these parameters to be passed from the WMS to the Compute Element; furthermore, this would also require all WMS administrators update their systems, and this is a task that requires coordination across the whole Grid.

2. The forwarded parameters are limited to existing JDL GLUE expressions;

3. It is not possible to tell if the Compute Element processes these requirements.

This method cannot be used to request resource types with complex requirements such as the GPGPU requirement above. A new method was developed that allows complex CDER requirements expressions to be added to the JDL, and processed by the LRMS.

Implementing CDER Independence

To describe how the new method works, a simple JDL (Listing 3.7) will be used to illustrate the implementation of LRMS independence.

Listing 3.7: Example Job Description in gLite JDL format for a job that requires two GPGPUs per worker node

```
1  [
2  Executable = "GPGPU_job_script.sh";
3  StdOutput = "std.out";
4  StdError = "std.err";
5  InputSandbox =  {
6                   "GPGPU_acquire_prologue.sh",
7                   "GPGPU_job_script.sh",
8                   "GPGPU_release_epilogue.sh"
9                  };
10 OutputSandbox = { "std.out","std.err"} ;
11 VirtualOrganisation = "gputestvo";
12 Requirements = (Member("NvidiaGPGPU",other.↩
      ↪ GlueHostApplicationSoftwareRunTimeEnvironment));}
13 GPGPUPerNode=2;
14 ]
```

The first item to notice (Line 12) is that the JDL requirement uses a GLUE 1.3 Tagged-Environment value "NvidiaGPGPU" as the basis for selecting Compute Elements that support Nvidia GPGPU CDERs. The Tagged-Environment is used here for simplicity, but a more sophisticated CDER selection process will be demonstrated in Section 3.3.5. Line 13 is the GPGPU requirement expression that needs to be translated into a LRMS directive. In this example, this expression follows JDL format and illustrates independence from any LRMS implementation.

The orchestration of the job through the Compute Element can be classified into three stages: (i) LRMS job preparation; (ii) job submission to the LRMS; and (iii) job execution on the selected worker nodes (Figure 3.6).

LRMS job preparation The *job preparation* stage is used here to convert the JDL GPGPU resource specification into an LRMS specification. During the Job Preparation phase, job input files and executable programs are transferred from the WMS (or UI, if direct job submission is used) to the CREAM CE. To aid execution of the job specified in the JDL, the CREAM CE has a "JobWrapper" service that converts JDL into a script that can be submitted to the local

LRMS. For example, this stage is used to convert common JDL directives, such as CPUNumber requirements, into local LRMS equivalents. The JobWrapper also adds support functions to transfer output files back to the WMS after the job has successfully run or report errors. Furthermore, the mechanism to enable the "Forwarding" of extra JDL requirement expressions into the LRMS is made possible by invoking an LRMS specific script (XXXX_local_submit_attributes.sh, where XXXX is the name of the family of LRMS used e.g. PBS). The output of this script is inserted into the JobWrapper script. In the prototype's case, the pbs_local_submit_attributes.sh can be used to inspect the JDL and convert the expression $GPGPUPerNode = X$; into an LRMS directive. The main modification to the middleware was the development of: (i) a function to retrieve the job JDL; and (ii) a JDL parser to return the value of a given key (e.g. $GPGPUPerNode$).

Job submission The JobWrapper is submitted to the LRMS as a job. This will only execute once the required resources are available.

Job Execution The job executes on the worker node(s) and transfers any JDL specified output files to the WMS. This stage is used in the prototype to implement GPGPU CDER exclusivity.

3.3.3 Exclusivity: Restricting Visibility of GPGPU Resources

Multi-core worker nodes allow many independent jobs to execute simultaneously. These independent jobs are protected from each other by executing in their own private disk-area, and by using file-access and process protection enforced by the operating system kernel. GPU (and consequently GPGPU) resources are designed to be accessible by all worker node users. In the case of GPGPUs exposed through a batch system, it is desirable to control access to them. Nvidia supports a number of ways in which access to its GPGPUs can be controlled:

1. the CUDA_VISIBLE_DEVICES environment variable can be used to restrict the visibility of a set of GPGPU devices in a process,

2. the Nvidia *Compute Modes* can permit/deny sharing of a GPGPU between multiple processes.

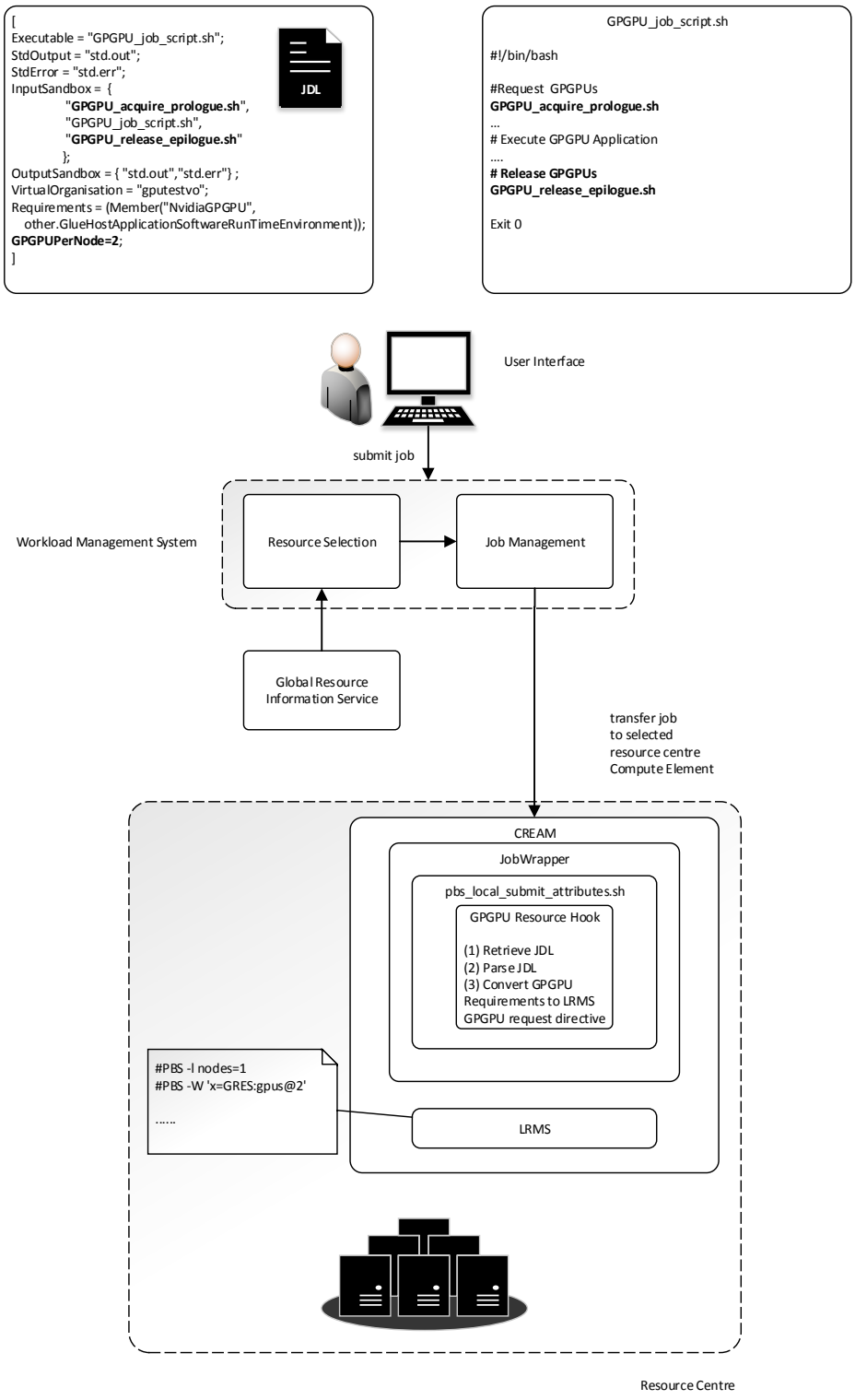


Figure 3.5: An Illustration of how the JDL and CREAM TORQUE/MAUI Compute Element are extended to support GPGPU CDERs. An extension converts the declaration **GPGPUPerNode=2** into a TORRQUE/MAUI GPGPU CDER request.

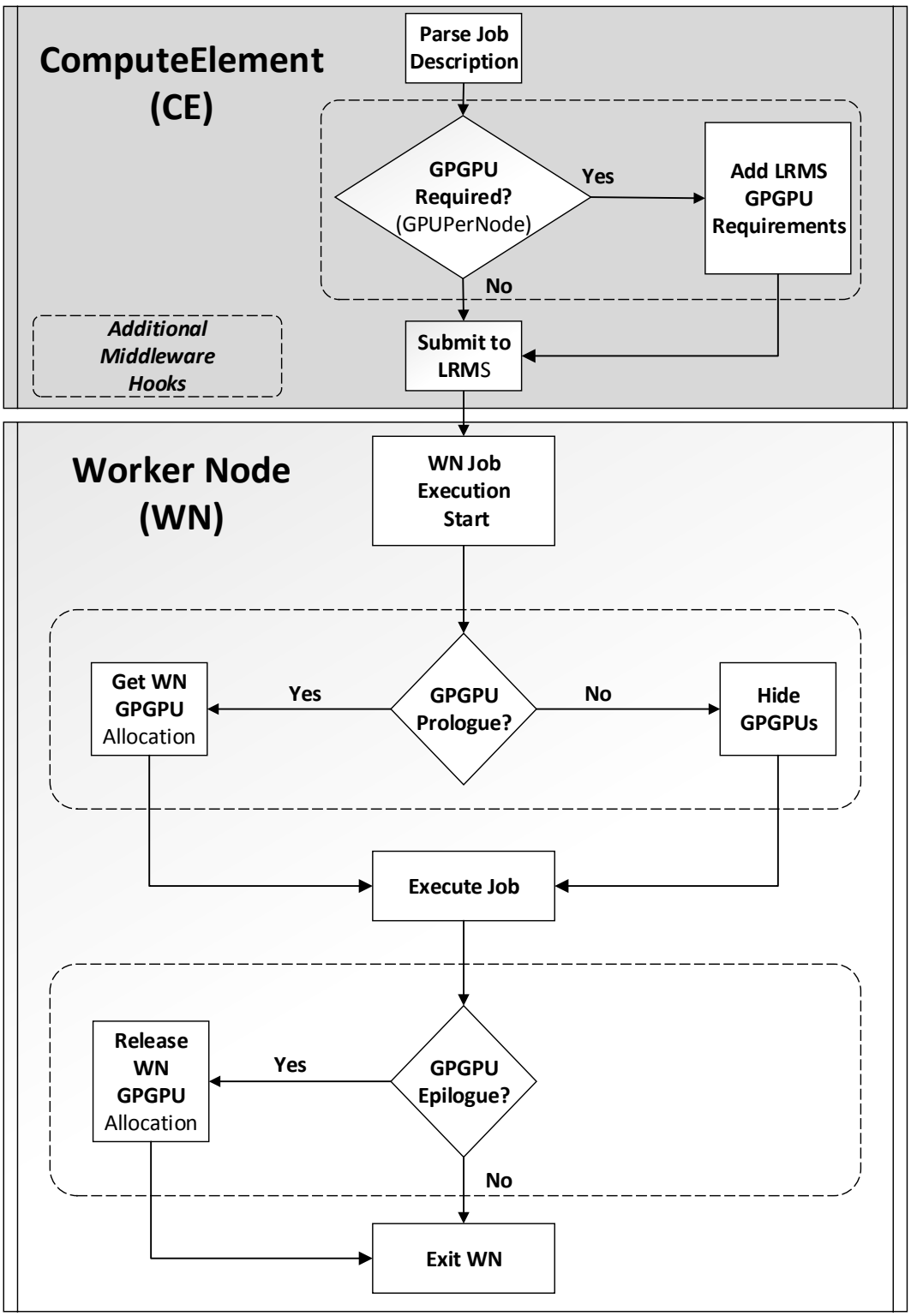


Figure 3.6: Batch System GPGPU allocation process.

The handling of GPGPU allocation on common open-source batch systems, such as TORQUE/MAUI, is still problematic, in particular with mid-range GPGPUs where Nvidia-SMI tools do not report a full complement of utilisation data. A simple prototype service was developed that allows TORQUE/MAUI batch jobs to request a set of free GPGPUs on the worker node, and to release those GPGPUs back to the service when no longer required. Furthermore, the allocated GPGPUs are not visible to other user jobs on the worker node.

The service, which executes on each worker node, implements a lightweight webserver using the Tornado [147] framework. To improve the security of the service, it runs as an unprivileged user, GPGPU allocation states are maintained in a lightweight persistent database, and the service is internal to the worker node (i.e. it is not available to other nodes).

The server responds to two types of requests: *requestGPUS* and *releaseGPGUs*. These requests are respectively executed from within the *GPGPU_acquire_prologue.sh* and *GPGPU_release_epilogue.sh* Job Hook scripts, specified in the JDL (Listing 3.7).

Requests to the prototype tornado server must adhere to a strict syntax, and all malformed requests are dropped by the server. In the case of the TORQUE implementation, a request is generated by sending a copy of the "PBS_JOBCOOKIE" and a list of *Universally Unique Identifiers* (UUIDs) - one per requested GPGPU. The PBS_JOBCOOKIE was chosen because, unlike the batch system "jobid", this value is not exposed to other users or processes.

The server manages the allocation of GPGPUs to jobs by using a single database with two tables: *UUID_JOBID* and *GPU_UUID*. The GPU_UUID table associates GPGPU devices with a job-generated UUID. The UUID_JOBID table associates UUIDs to a suitable unique value, such as the $PBS\_JOBCOOKIE$.

Initialisation

The GPU_UUID table is initialized at node start-up. A row is added for each physical GPGPU. Similarly, the UUID_JOBID table is created, but remains unpopulated until a GPGPU is requested by a job. Table 3.5 shows the initial state of the database tables for a node with two GPGPUs.

Table 3.5: Initial State

(a)

GPUID	UUID
0	
1	

(b)

JOBID	UUID

Allocation Request

A GPGPU allocation request (Figure 3.7) should be sent to the server before the job attempts to execute any GPGPU code. When the server receives an allocation request message, a "Request Handler" will attempt to validate it. If the request is valid, then the string is converted into two component values: a JOBID (PBS_JOBCOOKIE) and a list of UUIDs. The request handler then iterates over the list of UUIDS and adds (UUID,JOBID) tuples to the UUID_JOBID table. The handler will also iterate over the list of UUIDs, selecting the first free GPGPU (i.e. rows where the UUID cell is NULL) from the GPUID_UUID table. The selected row is the updated with a new (GPUID,UUID) tuple. Changes to the database are committed. Finally, the server returns a text string to the client in the form of a comma-separated list of integers. This is the list of GPGPU devices that the the job has been allocated. This value is assigned to a *read-only* CUDA_VISIBLE_DEVICE environment variable, thereby ensuring that the user or process can no longer (inadvertently) update its value.

(a) GPGPUs paired to UUIDs

GPUID	UUID
0	$UUID_1$
1	$UUID_2$

(b) $JOBID_1$ linked to $UUID_1, UUID_2$

JOBID	UUID
$JOBID_1$	$UUID_1$
$JOBID_1$	$UUID_2$

Table 3.6: Example of a single job running on a node with two GPGPUs

Release Request

A GPGPU "Release" request should be executed during the job's epilogue. This request is complementary to the "Allocation" request. As with the allocation request, the input is validated. If the request is valid, the server's "Request Handler" will remove all tuples matching the UUIDs from the UUID_JOBID table, and all the specified UUIDs from the GPUID_UUID table.

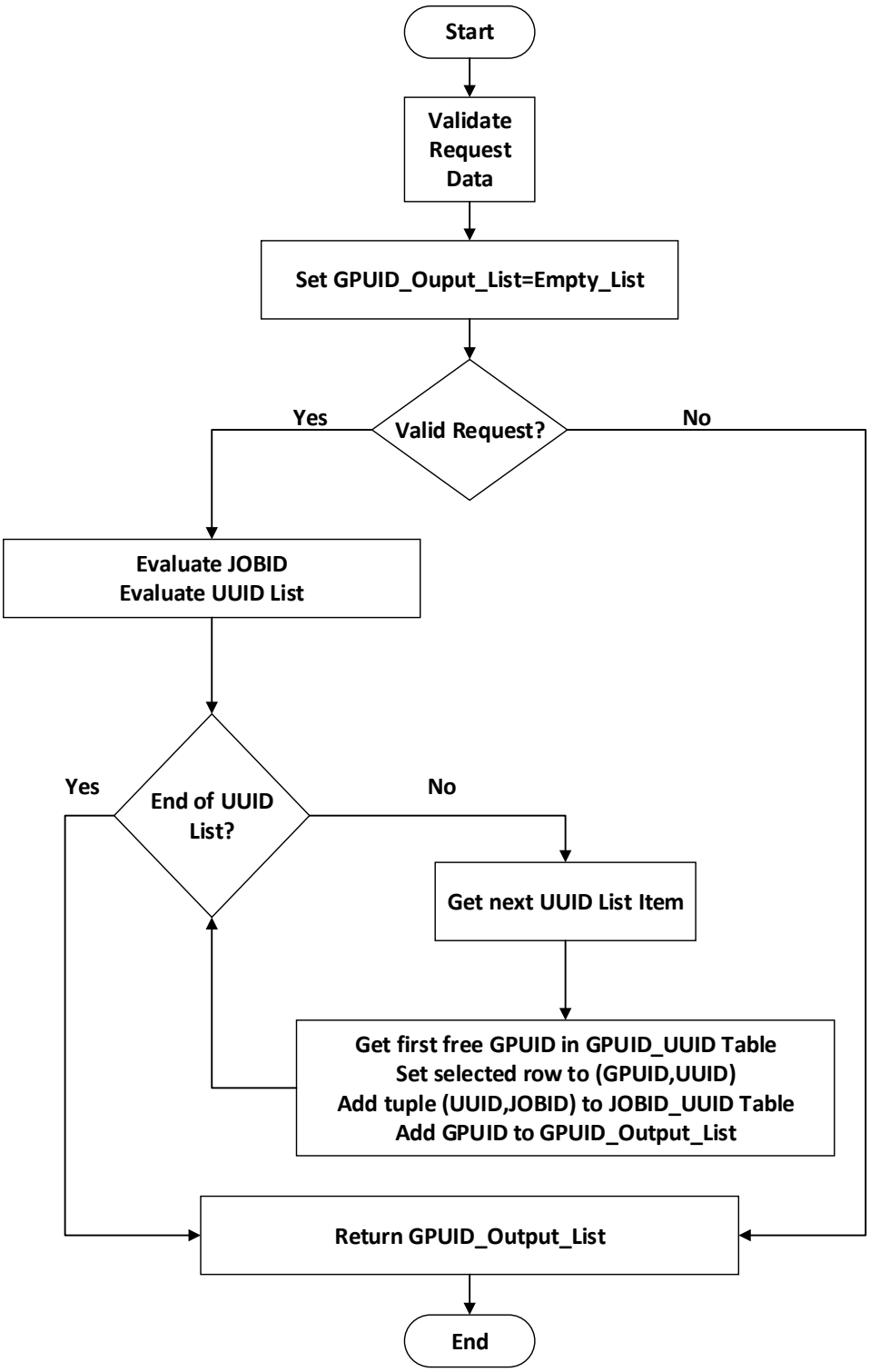


Figure 3.7: Worker Node GPGPU allocation subsystem

Token based access-control

The use of UUIDs as tokens in the GPGPU allocation and release process is partially based on the use of time-limited UUID tokens in the *Puppet* [148] fabric management system – a system used to install, configure and maintain machines on a network. In Puppet a token is used as a "shared-key" between the Puppet server and the client machine. The shared-key is used to authorise the download of the client's installation configuration file. Puppet ensures that the UUID shared-key tokens are expired (e.g. one hour after initial generation) to ensure that the tokens cannot be reused. In the case of the presented prototype, the lifetime of the UUID in the GPGPU allocation is limited to the duration of the job.

3.3.4 Direct Job Submission

Job submission was tested using different values for $GPGPUPerNode$. Sample output of a job that requested a single GPGPU but executed on a worker node with multiple GPGPUs is listed below (Listing 3.8):

Listing 3.8: Example GPGPU Job resricted to a single GPGPU

```
/usr/local/cuda/samples/1_Utilities/deviceQuery/deviceQuery
Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)
...
Detected 1 CUDA Capable device(s)
```

3.3.5 Two-Phase Job Submission

The example JDL in Listing 3.7 illustrated a new method to extend gLite's JDL capabilities. This extension allowed the JDL to include a requirement for GPGPU CDERs. However, this method on its own is not sufficient to discover or select Compute Elements based on more specific criteria, such as the model of GPGPU or its capabilities. To add this functionality, an implementation of the two-phase model (p. 83) is presented. The first phase is to match resource providers to a given GPGPU request (expressed as a JDL "requirement"). This phase acts as a filter that determines the set of Compute Elements matching the detailed GPGPU CDER resource requirements; the second constructs a new JDL that restricts the job to only those matching Compute Elements.

Phase-1

The basic operation of the two-phase model is to query all Compute Elements for the presence of a specific key/value pair that should be defined for the CDER. For example, this could be a query for the GLUE 2.0 ExecutionEnvironment attribute GLUE2EntityOtherInfo="GPGPUVendor=Nvidia", or from Listing 3.7, the GLUE 1.3 GlueHostApplicationSoftwareRunTimeEnvironment="NvidiaGPGPU". For each matching ComputeElement a simple parser extracts the CDER data and converts it into a ClassAd Machine Resource Offer. An example conversion of the GLUE2 data in Listing 3.4 would yield the ClassAd Resource Offer shown in Listing 3.9.

Listing 3.9: ClassAd Resource Offer derived from Listing 3.4

```
MyType = "Machine"
Machine = "wn140.grid.cs.tcd.ie"
GPGPUTotalInstances = 32
GPGPUUsedInstances = 2
GPGPUCUDAComputeCapability = 2.1
GPGPUMainMemorySize = 1024
GPGPUMP = 4
GPGPUCoresPerMP = 48
GPGPUCores = 192
GPGPUClockSpeed = 1660
GPGPUECCSupport = false
GPGPUVendor = "Nvidia"
GPGPUModel = "GTS_450"
GPGPUPerNode = 2
Requirements = true
Rank = 1
```

The user-defined JDL GPGPU requirements in Listing 3.10 illustrate a ClassAd Resource Request. The ClassAd Machine Resource Offer can be matched against the request. If a match is met, the name of that Compute Element can be stored for further processing.

Listing 3.10: Example JDL using GPGPU attributes

```
Requirements = GPGPUVendor=="Nvidia" && (GPGPUMainMemorySize>=512);
Rank = 1;
```

Phase-2

The second phase begins by processing the list of matching Compute Elements. The result of this processing is the generation of a new JDL requirements clause (a partial requirements statement). That new clause can be passed into a template JDL to create a new JDL. This new JDL is submitted to the Grid through the WMS.

Listing 3.11: Example JDL using GPGPU attributes

```
[
Executable = "GPGPU_job_script.sh";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox =  {
                  "GPGPU_acquire_prologue.sh",
                  "GPGPU_job_script.sh",
                  "GPGPU_release_epilogue.sh"
                };
OutputSandbox = { "std.out","std.err"};
VirtualOrganisation = "gputestvo";
# Example CELIST=RegExp("wn140.grid.cs.tcd.ie*", other.↩
   ↪ GlueCEUniqueID)
CELIST=<%= GPU_CE_LIST %>;
Requirements = (CELIST && Member("NvidiaGPGPU",other.↩
   ↪ GlueHostApplicationSoftwareRunTimeEnvironment));}
GPGPUPerNode=2;
]
```

3.4 Other Related Work

Hierarchical Brokering

Toor et al [149] describe a prototype extension to the ARC middleware's job-submission system on a User Interface. Additional static and dynamic information about resources are published through a GLUE 2.0 XML-based information system. ARC performs brokering at the User Interface, and this is known as *Distributed Brokering*.

Under the gLite/UMD middleware, job submission can either be directed at a user-specified resource without any brokering or left to a job-orchestration system (such as the Workload Management System). The WMS is an example of *Centralised Broker-*

ing. The two-phase method presented in this thesis implements *Hierarchical Brokering* with the pre-filter stage acting as a Distributed Broker and the final job-orchestration through the WMS acting as a Centralised Broker.

The available brokering criteria (Random, FastestQueue, Benchmark, Data) [90] used by the ARC middleware to select resources is not as comprehensive as those available under the WMS. Indeed, the WMS match-making system simultaneously supports multiple criteria (Queue Utilization, Benchmark, etc.) for ranking matching resources.

CUDA_wrapper

CUDA_wrapper [150][151] was developed to help facilitate secure and controlled access to Nvidia-based GPGPU resources on multi-user GPGPU clusters. The wrapper acts by intercepting normal CUDA API fuction calls and overloading them with additional methods that transparently provide additional key-based access-control to the raw GPGPU device.

While the CUDA_wrapper initially looked like a good solution to handle GPGPU resource management under TORQUE, further investigation revealed several weaknesses, namely: each CUDA_wrapper API interception imposes additional latency to handle each CUDA function call; CUDA_wrapper must be recompiled for each new version of CUDA; CUDA_wrapper did not work under testing with CUDA 5.5; and, the wrapper is vendor and LRMS (TORQUE) specific. The approach presented in this chapter LRMS avoids these pitfalls.

Application Hook Method

The design and implementation of the presented execution model used in this thesis follows a pattern similar to that used by MPI-Start [152]. In particular, this is evident in how software hooks are used to build an appropriate site-local application runtime-environment. The major differences in the implementation are: (i) MPI enabled resource centres currently publish information about the local MPI environment as a set of GLUE 1.3 SoftwareEnvironment tags. This information is inefficiently converted into a set of discrete and seemingly unrelated GLUE 2.0 ApplicationEnvironment entities; (ii) The GPGPU implementation attempts to exploit many of the newer features of the GLUE 2.0 ApplicationEnvironment definition. This includes the

ability to publish GPGPU resource capacity and utilisation information.

Other Information Systems

The Relational Grid Monitoring Architecture (R-GMA) [109] was developed as a generic grid information system based on a Producer/Consumer model. Inserting, updating, deleting, and querying of data was available through command-line tools and an API that supported a subset of the SQL92 query language standard. In addition to publishing GLUE 1.3, R-GMA allowed users and services to publish, consume, update, and query other non-GLUE data. This system was used to publish LRMS Usage Records to monitor the usage of Compute Resources across the EGEE, a grid infrastructure that evolved into the current EGI. User-based applications included grid-wide intrusion detection [153]. The R-GMA may have been a suitable facility for Sites to publish up-to-date information about CDERs without depending on the GLUE and the GIS; however, this is no longer in use anywhere.

Handling Software Licenses

The question of how to handle limited software licenses in grid environments is quite similar to the CDER problem, particularly in the case where access to software is *node-locked*; i.e., the software may only run on certain nodes due to license restrictions. A typical way to handle this in the LRMS is to assign a nominal "property" to the node – when the user needs a license, then a specific LRMS directive ensures the allocation of only those nodes satisfying this property. Furthermore, by treating the software as a generic consumable resource (which is now supported by most LRMS's), the usage of the software can also be managed by the LRMS. The key differences between the treatment of CDERs and the Software Applications is that the GLUE 2.0 ApplicationEnvironment can already be used to publish the availability of particular software as well as to indicate capacity and usage of licenses. Other GLUE 2.0 Entities such as the ApplicationEnviromentHandle can be used to guide the user application on how to configure or bootstrap the application's environment.

3.5 Summary and Conclusions

This chapter investigated the challenging problem of integrating additional non-CPU resources (e.g. GPGPUs) into Grid infrastructures. Instead of approaching the integration of new resources on a case-by-case basis, an abstraction of the resource integration problem was conceived. A proof-of-concept solution to the integration of GPGPU accelerators was then applied.

Large eScience-driven Grids must offer a reliable and stable quality of service to potentially many thousands of users over an extended and continuous period of time (ranging from days to months to years). The challenge arises because current Grids need to be more flexible in how they integrate new technologies, whilst at the same time, ensuring stability. Major changes to the Grids (e.g. GLUE, middleware and user applications) are not desired. The approach taken here is to investigate if there are existing provisions within Grids that may be leveraged to integrate arbitrary LRMS resources.

The abstract approach looks at the higher-level requirements needed to integrate many different types of resources. This approach is fundamentally different to contemporary integration solutions that focus on integrating single specific resource types (e.g. GPGPUs). The abstract approach in this thesis proposed establishing a set of fundamental restrictions on what resources should be considered and how the resources could be used to execute user applications. These restrictions require that the collection of resources are finite, and that there must be some mechanism to track or account for their allocation. There are further limitations that require resources to be bound to particular worker nodes, and that they are exclusively allocated to a single job. Such resources are called *CDERs*. The abstraction is intended to capture the finite nature of hardware resources and to help impose some control on their use (by avoiding resource allocation contention).

To help fully integrate these CDERs into the Grid, five integration requirements (principles) were considered. However, only three of those requirements are related to how a resource is used in a grid job. These three properties are called: (i) Discovery – resource properties can be represented/published in the Grid Information System; (ii) Independence – the grid middleware must support a Job Submission method to specify the resource in a job specification language, and this specification must be independent of specific LRMSs; and (iii) Exclusivity – the resource must be under the

control of a unique single job. The other two requirements, Accounting and Monitoring, have no bearing on how grid jobs are handled, so were considered outside the scope of this thesis.

To address Discovery and Job Submission, five separate conceptual strategies that handle CDERs were considered. Each strategy was evaluated by considering: (i) whether it supported resource discovery; (ii) if it could publish extensive details about the resource; (iii) how each of the resource properties could be related to one another; (iv) whether dynamic updates to the data were possible; (v) what GLUE version it supported; (vi) how long it took to query and collate the information for a given resource; (vii) what was the overhead in publishing the resource data; and (viii) whether current job submission techniques work, or if they must be handled by another means (e.g. the two-phase job submission).

Any strategy for integrating CPU-dependent execution resources into grid infrastructures must support the fine-grained publication of the resource's properties and capabilities. In addition, the dynamic publication of the number of such resources installed (capacity), their utilisation, and fair share policies are also required. Complementary mechanisms to discover and select resources based on these factors (properties, capabilities, capacity, utilisation, and fair shares) allow grid users to refine the selection of resource providers to only those that match their needs.

Of the five conceptual strategies investigated in this chapter, some strategies that have previously been applied to grid – A Priori, Named-Queue, and Tagged-Extension – cannot be effectively used with CDERs. The final two strategies – Attribute-Extension and Class-Extension – have been shown to be the most-flexible strategies for adding arbitrary (CDER) attributes and their values. However, they have their own weaknesses and strengths. The Attribute-Extension strategy is more data efficient, and the extension data is added internally to the object instance. The Class-Extension strategy creates new Extension instances, and this incurs a data volume penalty. However, updating individual attribute values will be more time efficient.

Although the match-making time measurements (Table 3.4) show that the Class-Extension strategy is slightly slower than the Attribute-Extension strategy, there is scope for reducing the time taken for Class-Extension even further. The measured time includes the cost of retrieving *all* Key/Value pairs for the GPGPU CDER. This need not be the case for the Class-Extension strategy. Only the Keys/Value pairs corresponding to the keys specified in the job-requirements expression are needed when

generating the Resource Offer. Under Attribute-Extension, all OtherInfo values are retrieved – this is due to limitations in constructing more-specific LDAP queries.

The conceptual approach was applied to the integration of GPGPU resources. It was shown how a GLUE 2.0 based multi-discipline scientific grid can be extended to support applications that accessed GPGPUs. A methodology was developed that applied three resource integration principles, namely: *Discovery*, *Independence* and *Exclusivity*.

The presented prototype is one of the first examples of where a GLUE 2.0 ExecutionEnvironment entity has been extended to include additional attributes that describe the capacity, utilisation and other properties of hardware used directly by the application itself. These attributes can be generated either statically or dynamically. The example use-case demonstrates publishing the total number of installed Nvidia GPGPUs, their current utilisation, and some selected hardware properties. This conforms to the *Discovery* principle.

A method was developed that allows a grid user to specify very fine-grained GPGPU requirements in the Job Description Language. For example, the prototype allows the user to specify the number of GPGPUs required per worker node, and to choose only GPGPU resources that match a minimum amount of memory. The job requirement is converted into the native LRMS resource specification once the job is submitted to a Compute Element. This is an application of the *Independence* principle.

Ensuring that users have guaranteed and isolated access to GPGPUs in a multi-user system can be difficult. This problem is compounded in the cases where multiple GPGPUs on the same worker node can be accessed by multiple users - some LRMSs (in particular TORQUE/MAUI) do not indicate what GPGPU has been assigned to each user job. The worker node GPGPU access control system discussed in Section 3.3.3 can assuage this problem. The development of a service to manage GPGPU allocation on these systems ensures the *Exclusivity* principle.

The application of the conceptual approach to GPGPU integration showed that it was possible to quickly add significant value to the capabilities of existing Grids – it produced a methodology to rapidly integrate new resources. These resources can advertise very fine details about their properties and capability. This integration is achieved without having to adapt the GLUE Schema (time-consuming) and without making major changes to the grid middleware (time-consuming and prone to introducing middleware bugs).

Chapter 4

Grid-enabled Virtual GPGPUs

This chapter describes a new model for the execution of virtual GPGPU-enabled applications based on commonly used application frameworks such as CUDA and OpenCL. While this model is intended for exploitation on grid infrastructures, a particular strand of this work, the LRMS integration of virtual GPGPUs (vGPGPUs), can be considered independently of Grid, and is applicable to non-grid clusters, including High Performance Computing and High Throughput Computing environments.

The starting point of this work is the development of an abstract architecture for the construction of vGPGPU resources. This is expanded to an abstract architecture for LRMS integration that facilitates vGPGPU resource requests and resource allocation management. The next step is the development of a concrete implementation, i.e. a system that constructs vGPGPUs, and allows them to be treated as LRMS resources. The abstract architecture is further developed on the basis of the Grid Resource Integration Principles (p. 76), although the principles are modified to handle virtual resources. These principles are applied to the concrete LRMS implementations to support vGPGPU discovery and job specification on the Grid.

From a user's and resource provider's perspective, the advantages of vGPGPUs are the following: (i) they provide a flexible and powerful mechanism for specifying the number (and type) of GPGPUs required by a job; (ii) unlike the work presented in Chapter 3, the job is not restricted to resource providers that satisfy the ability to allocate a minimum number of physical GPGPUs on each worker node; (iii) instead, the allocation of GPGPUs to the job is based on the availability of GPGPU resources from a central pool; (iv) more importantly, it allows a job to transparently use more GPGPUs than are available on any single worker node using a single API (e.g. CUDA

or OpenCL). From a resource provider's perspective, this new arrangement provides greater flexibility in the allocation of GPGPUs to jobs; and (v) it has the potential to increase GPGPU resource utilisation and job throughput. Virtual resources can be allocated to jobs running on other worker nodes, even when the node with the physical resource is fully utilised.

Central to the model is the use of multiple virtualisation layers to create a "frontend/backend" architecture to facilitate physical separation between the worker node upon which a job executes (CPUs) and the node that hosts the GPGPUs used by the job. This particular architecture will enable reconfigurable access to many vGPGPUs from any worker node. Furthermore, through the use of existing technologies, it will be possible to transparently access non-local (backend) GPGPUs within applications that execute on the "frontend" worker nodes.

Section 4.1 presents the abstract frontend/backend vGPGPU model. Section 4.2 looks at applying this model to several LRMS implementations. This work includes building new services to handle vGPGPU resource management. Two case-studies showing the integration of vGPGPUs into some popular LRMSs are presented.

It will be shown in Section 4.3 how the Grid Resource Integration Principles (p. 76) can be applied to these new vGPGPU resources to allow full grid integration – the vGPGPU will satisfy the resource discovery, independence and exclusivity requirements. Furthermore, it will be achieved by using existing middleware, which is adapted by making only minimal changes to it.

Section 4.4 uses several benchmarks (p.32) to examine the performance of the model, examining the performance of the vGPGPU implementation and the job throughput performance in comparison to jobs that use non-virtualised physical GPGPU (pGPGPUs). Section 4.5 reviews other related work. Finally Section 4.6 summarises the presented work.

As noted above, the proposed approach builds on existing virtualisation software to provide access to vGPGPUs. The specific contribution of this work is the construction of a model that consists of:

(a) a new vGPGPU Factory service to orchestrate the creation of vGPGPU VMs;

(b) an external vGPGPU resource management service that enhances the existing LRMS and provides vGPGPU resource allocation where no such support exists in the LRMS;

(c) support for multi-layered GPGPU virtualisation. This approach allows a more flexible use of GPGPU resources, and is capable of supporting several common GPGPU application interfaces; and

(d) a grid-layer that enables the use of virtual-GPGPUs on existing grids.

4.1 Abstract Virtual GPGPU Architecture

This section begins by exploring vGPGPUs in the context of a set of worker nodes managed by an LRMS (i.e. cluster integration). This is a necessary step before they can be fully integrated into the Grid itself. Full grid integration will be explored in Section 4.3.

As discussed in Chapter 3, a typical resource, such as a GPGPU, is physically fixed to the worker node on which it is installed. If a user requires a GPGPU, then it must be specified in the job description. After the job is submitted, the LRMS will hold the job in a waiting state until the requested number of CPUs and GPGPUs are available on each worker node. The job will be allowed to execute only when the resources have been assigned to it.

One possible source of resource allocation and usage inefficiency is that the LRMS may fill a GPGPU worker node with non-GPGPU jobs. In particular, if an LRMS is processing many jobs, some GPGPU jobs may be left waiting longer than should be ordinarily expected because non-GPGPU jobs have been assigned to worker nodes with GPGPUs. By steering non-GPGPU jobs to non-GPGPU worker nodes and by reserving CPUs for GPGPU jobs on GPGPU worker nodes, it may be possible to reduce the amount of time that a job must wait for a GPGPU. However, an alternative approach is to "decouple" the physical GPGPU resource from the worker node and allow that resource to be accessed from any worker node. Such decoupling is made possible through resource virtualisation.

As noted in Section 2.3 OS-level, Full- and Para-virtualisation are interesting machine virtualisation models because, when used with GPGPU virtualisation techniques, they provide the ability to assign GPGPU resources to a VM[1]. However, the VM is still limited to accessing only those GPGPU resources that are present on the physical machine. To build VMs that can access more GPGPU resources than are physically

[1] A physical machine in the case of PCI-switching

present requires an additional layer of virtualisation.

The approach taken here will be to identify each (abstract) subsystem and the role, requirements and interactions that they have with each other. The implementation details are not defined at this stage, so as to ensure that the architecture is technology agnostic.

Like machine virtualisation, several approaches to providing virtual GPGPUs (p.39) have also been implemented. It is also possible to combine multiple layers of machine virtualisation and GPGPU virtualisation to construct new and interesting computer architectures which can provide services that are not possible with a single virtualisation technology. For example, it is possible to combine PCI-switching with PCI-passthrough, and it is also possible to combine OS-level virtualisation with API-Interception. In both cases, the combination of GPGPU virtualisation techniques can produce a machine (physical or virtual) that has access to many GPGPUs as if they were locally attached. While the former is an expensive solution that could be beneficial to Cloud Computing resource providers, the latter is a lower-cost solution that can be used by HPC, HTC and Grid resource providers.

The use of a multi-layered vGPGPU model is proposed in this thesis. This model is based around having one layer that is able to address each GPGPU atomically (in isolation to the other resources), and then a second layer that allows the individual GPGPUs to be combined together and assigned to a VM (or indeed, as with PCI-switching, a physical machine). An isolated GPGPU will be referred to as “backend”, while the VM that accesses them is called a ‘frontend”.

This model is mediated by a new service, the *Registry*, that facilitates vGPGPU resource management. Resource requirements are specified in the LRMS native job description language. The LRMS can process these requirements (e.g request floating consumables from the LRMS) and propagate further directives to the worker nodes (e.g. information on the number of resources to request from the Registry or the information about the runtime environment needed to execute the vGPGPU job). The abstract components and services are identified in Figure 4.1.

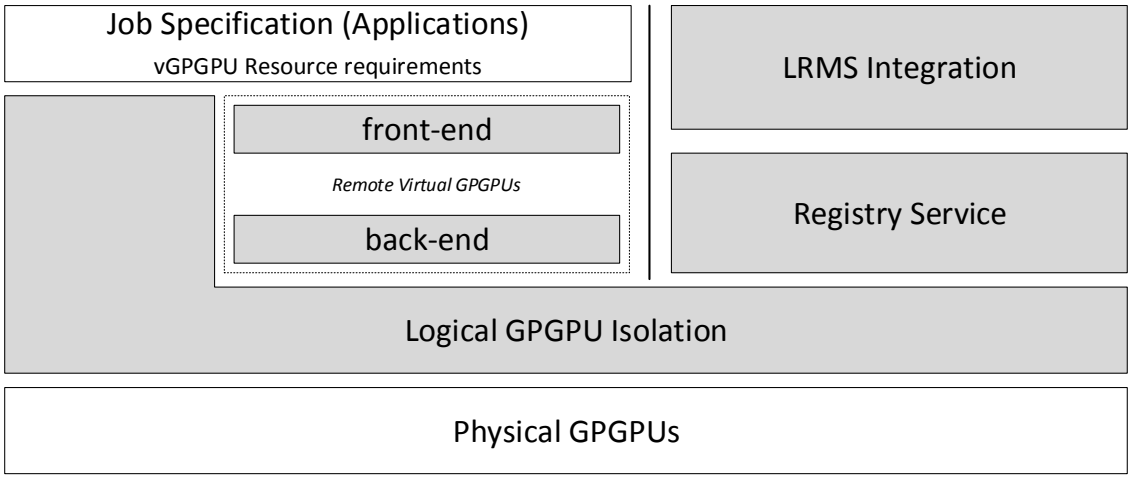


Figure 4.1: An component-based view of the abstract model for vGPGPU LRMS Integration

Logical GPGPU Isolation

As noted in Section 2.3.3, it is typical for Resource Centres (Grid and Clouds) to consolidate as much hardware capacity and capability into the smallest feasible number of physical machines. With appropriate hardware, GPGPUs can be densely packed into these machines. Without GPGPU virtualisation or additional access controls, however, user jobs will have access to all of the co-located GPGPUs – controlling resource allocation and resource access is a challenging problem in multi-user environments.

An approach that logically assigns each GPGPU to its own controlling virtual machine or lightweight VM, using a virtualisation technique such as *Kernel Device Passthrough* or *PCI-Passthrough*, can help avoid this problem. Fine grained access to the Logical GPGPUs can by achieved by assigning individual users to VMs that have been preconfigured with access to one or more GPGPUs.

Under this model, workloads that require a single GPGPU can be accommodated by assigning a single VM to the job, giving it access to one (or more) CPUs and one GPGPU. To accommodate multi-GPGPU workloads, applications may use message passing techniques (e.g. MPI) to transfer data between VMs (and their GPGPUs), and to coordinate execution.

Logical GPGPU Isolation, by itself, cannot accommodate multi-CPU/multi-GPGPU applications. For example, workloads that require 4 CPUs and 8 GPGPUs cannot be handled. It does, however, provide the foundation for a model with multi-CPU/multi-GPU capability when used in conjunction with a remote GPGPU access technology, as

proposed below. The vGPGPU framework developed in this thesis aims to construct worker node environments that support multi-CPU/multi-GPGPU applications.

Remote Virtual GPGPUs (vGPGPUs)

By using a GPGPU virtualisation technology capable of providing remote access to GPGPUs, a resource centre can provide user jobs with access to logically isolated GPGPUs. This allows the GPGPUs to be considered as a pool of *floating* consumable resources, referred to here as *vGPGPUs*. Under this model, the location of the *vGPGPU* allocated to a job is independent of the job's actual worker node. Furthermore, the model permits access to a greater number of GPGPUs than may be available on any single worker node, thereby facilitating multi-CPU/multi-GPGPU execution models. The model presented later in this chapter focusses on the use of API-Interception to access remote GPGPUs. An alternative (high-performance but high-cost) approach, however, could be based on PCI-Switching.

vGPGPU Registry Service

The vGPGPU resources provided by a resource provider are *pooled* together and placed under the control of a *Registry Service*. The Registry Service acts as a broker, managing a pool of vGPGPUs and the allocation of one or more vGPGPUs to user jobs. The principal functions of the Registry Service can be classified as:

(a) to allow the existence of vGPGPUs to be registered after initial creation on the physical hardware;

(b) to allow jobs to request vGPGPUs from the Resource Pool; and

(c) to allow the LRMS or other services to determine capacity and resource utilisation.

LRMS Integration

The allocation of the frontend nodes required for a particular job is handled by the LRMS as part of the normal job orchestration procedure. The number of nodes required is calculated based on the number of CPUs and the number of CPUs per worker node (SMP granularity) requested by the job.

The assignment of backend vGPGPU nodes to each frontend is facilitated by the vGPGPU Registry Service. First, however, a mechanism is needed to block jobs from

executing if sufficient GPGPU resources are not currently available. If the LRMS in use provides a *counting semaphore* facility for limiting access to resources (such as floating software licenses), then a counting semaphore can be initialised with the number of backend vGPGPUs. When a job is submitted to the LRMS, the job requests that the counting semaphore be decreased by the number of required vGPGPUs. As long as the value of the counting semaphore is greater than or equal to the number of vGPGPUs requested (and all other job requirements are satisfied) the job will execute, otherwise it will be blocked until resources become available. (An alternative to the use of an LRMS counting semaphore is also proposed later in Section 4.3.3.)

Once the job is running on its assigned worker node, it requests the required backend nodes from the Registry Service. It is the information provided by the Registry Service at this stage that associates each frontend node with specific backend nodes. When the job is finished, the requested resources are returned back to the Registry Service and the counting semaphore is increased.

Job Resource Specification

GPGPU applications are not uniform in their resource needs. The abstract model requires that there is a means to specify the required number of GPGPUs for single-CPU/single-GPGPU, single-CPU/multi-GPGPU and multi-CPU/multi-GPGPU applications. It may be desirable to be able to identify particular APIs or the type of GPGPU hardware that can be utilised by the application, or minimum requirements, e.g. for memory capacity.

4.2 vGPGPUs as LRMS Resources

The abstract model assumes that it is possible to create a pool of atomic/isolated backend resources, and these are assigned to frontend worker nodes by way of an intermediary service called the Registry. The model also requires that there is a basic set of vGPGPU resource requirements that the user must be able to specify (e.g number of vGPGPUs required and the runtime environment needed), and that these requirements can be propagated through the LRMS and into the frontend (if needed). The LRMS can interact with the Registry to help build a runtime environment according to those requirements.

A realisation of the abstract model is proposed that uses both OS-level virtualisation to create the backend resources and API-Interception to manage GPGPU access between the frontend and the backends. A new set of services will be introduced to add the missing backend resource management capabilities in accordance to the abstract model presented above. It should be noted that LRMS integration of vGPGPUs using API-Interception is not new, indeed, there is already prior art that integrates both rCUDA and VCL into SLURM – albeit, these both implement separate solutions. Furthermore, both suffer from some significant weaknesses, including:

(i) they are SLURM specific; and

(ii) if both rCUDA and VCL solutions are integrated with SLURM, then both attempt to manage the same vGPGPUs independently of each other. This independent resource management has the potential for resource management conflicts and may yield inconsistent views of actual resource states (e.g VCL/SLURM may try to assign a resource that rCUDA/SLURM has already claimed).

The prototype seeks to address these problems, and will do so by decoupling the backend resource management from specific LRMS/API-Interception implementations. Instead, their management is given over to a (new) single external subsystem. Interaction between the LRMS and this subsystem will help control backend allocation and deallocation requests, and this is facilitated through new LRMS "prolog" and "epilogue" scripts. The use of LRMS prolog and epilogue scripts to transparently execute extra instructions is inspired by ViBatch [154]. These vGPGPUs should be easy to use, with much of the complexity hidden from the user. This method is analogous to the method employed in Section 3.3. To aid this, a simple set of new key/value job attributes are supported, namely: the number of nodes (CPU cores), the number of vGPGPUs per node, and the type of API-Interception used by the job.

This basic model, which will be shown to be applicable to a variety of different LRMSs, consists of three new component parts:

(a) the *vGPGPU Factory* subsystem creates backend VMs;

(b) the Registry, which is used to aid both the installation and management of the backend VMs; and

(c) a set of LRMS-specific script-based plugins that act as a bridge between the user's vGPGPU job, the LRMS, and the Registry.

4.2.1 The vGPGPU Factory – Logical GPGPU Isolation

The vGPGPU Factory is a service that manages the life cycle of isolated backend VMs. The service starts by examining the node's hardware profile. Several properties (e.g. the GPGPU OS device name/number, the device vendor) are evaluated when a GPGPU is found. If a backend VM does not already exist, then a new one is constructed and labelled with a *Universally Unique Identifier* (UUID).

The UUIDs are derived either directly from the GPGPU hardware (Nvidia), or constructed from a combination of the physical machine's fully qualified domain name (FQDN) and the GPGPU's OS device name (AMD). The uniqueness of the UUID is used to bind a backend VM to the GPGPU, and is used for both resource management and to manage VM persistence across reboots of the hosting machine. It is more time efficient to reuse an existing backend VM than create a new image every time a job needs it. By reusing the existing backend VMs, the time taken to start them is significantly reduced.

To ensure that each backend VM is restricted to a given hardware device (i.e. logically isolated), the Nvidia and AMD runtime environment variables (CUDA_VISIBLE_DEVICES or GPU_DEVICE_ORDINAL respectively) are set to the GPGPU's device number. Variables and GPGPU devices are passed into the VM at construction time.

Finally, the GPGPU vendor value is used to determine which API-Interception software is installed and started on the VM. This is VCL for all Nvidia- and AMD-based VMs, and rCUDA for Nvidia-based VMs. The construction also ensures that multiple API-Interception virtualisation stacks are supported according to the hardware type.

Docker [155] is used to build the VM and to install both the rCUDA/VCL software and prepare the VM environment. In addition, network bridging using Pipework [156] is used in preference to Docker's native Network Address Translation (NAT) solution because rCUDA did not function correctly under NAT, and because Pipework allows IP address assignment to the VM. In this way the VM's IP address can be managed through the Registry and assigned to the VM when it is initially created or instantiated. An algorithm for the Factory VM build process is outlined in Algorithm 1.

```
GPGPUVendors = {Nvidia, ATI};
foreach Vendor in GPGPUVendors do
    VendorDeviceCount=getVendorDeviceCount(Vendor);
    for DeviceID=0 to VendorDeviceCount − 1 do
        UUID=getDeviceUUID(Vendor, DeviceID);
        containerIP=RegistryGetIP(UUID);
        CreateContainer(Vendor,DeviceID,UUID,containerIP);
    end
end
```

Algorithm 1: Algorithm for Managing vGPGPUs using the GPGPU Container Factory Service

4.2.2 vGPGPU Registry Service

The Registry service is an extension of the worker node GPGPU resource management service introduced in Section 3.3.3 to a service that manages virtual GPGPU resources across an entire cluster. The Registry augments the resource management provided by the LRMS. Without the Registry the set of backend VMs created by each Factory form a pool of disjoint and unmanaged resources, i.e the resources exist, but their allocation is not coordinated. To add management capabilities, a new web-based service, the *vGPGPU Registry Service* (or *Registry*), has been developed. This service helps manage two aspects of backend VMs: their life cycle and their allocation to jobs. The service implements a simple interface (Figure 4.2) that interacts with a persistent database, and this database maintains the state information for each backend VM. The protocol and database schema are designed to be independent of LRMS implementations for portability across cluster systems. The operations can be divided into two categories:

(i) those operations to add or remove backend VMs from the Registry (performed by the Factory service); and

(ii) those operations used to request backend VMs during the execution of a job (performed by a frontend, i.e. a worker node).

In this prototype implementation the Registry interface is implemented using the HTTP protocol.

4.2.3 LRMS Integration and Job Specification

The LRMS plugin component is the only subsystem that requires specific customisation. Two LRMS use-cases demonstrate this integration: (i) SLURM; and (ii) TORQUE/-

Method	Description
register	Registers a new vGPGPU container that is being added to the pool. **Input**: Container UUID, GPU Vendor, GPU Device Number; **Output**: IP address of Container
unregister	Unregisters a vGPGPU container that is being removed from the pool. **Input**:Container UUID; **Output**: None.
request	Request a vGPGPU allocation for a job. The Registry returns the information needed to construct the vGPGPU front-end. **Input**: JobID, Number of vGPGPUs, Node Number, API-Interception Method; **Output**: List of vGPGPU Container IP addresses.
release	Release a vGPGPU allocation when it is no longer required by a job. The released vGPGPUs become available for allocation to another job. **Input**: JobID; **Output**: None.
query	Return information such as the number of free vGPGPUs and the number of vGPGPUs supporting a specified API (e.g. CUDA, OpenCL). **Input**: QueryName, API-Interception Method; **Output**: Integer.

Figure 4.2: vGPGPU Registry Service Interface

MAUI. The key differences between the LRMSs affect how vGPGPU jobs are handled and scheduled – these include support for non-CPU resources, and how arbitrary job parameters are propagated into the job's execution environment. However, despite these differences, vGPGPU jobs depend upon three LRMS-independent factors:

(i) the number of frontend nodes;

(ii) the number of backend VMs required by each frontend; and,

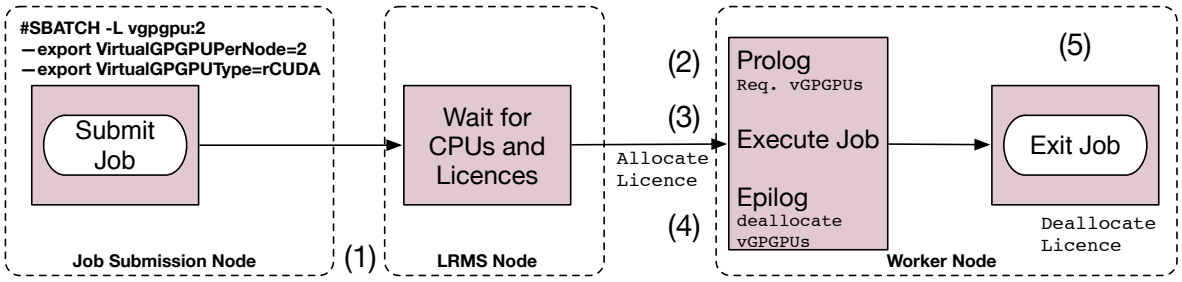
(iii) the API-Interception to be used.

The prototype assumes a natural mapping between an LRMS node – which in practice is a CPU core – and a vGPGPU frontend node. This implies that the number of frontend nodes is specified by declaring the number of nodes (or cores) required. LRMS environment variables are used to define the number of backend VMs (*VirtualGPGPUPerNode*) and the API-Interception required (*VirtualGPGPUType*). These can be passed from the LRMS to the frontend worker node (WN), where they are used by job prolog and epilogue scripts. It should be noted that the names of these LRMS environment variables are entirely arbitrary. The scripts transparently hide the complexity of configuring the vGPGPU job environment and interact with the Registry to allocate backend VMs. The manner in which resource requirements are passed into the LRMS and processed at runtime on the worker nodes by using prolog and epilogue scripts mirrors the technique used in Section 3.3.2.

Use-case 1: SLURM

The flow of a SLURM-based vGPGPU job is illustrated in Figure 4.3. In Step (1) the number of frontend nodes is specified by requesting normal SLURM nodes; backend VMs are specified by requesting one or more *vgpgpu* licences; and the VirtualGPGPUPerNode and VirtualGPGPUType variables are also *exported* to the WN environment. The job will remain in a waiting state until the specified number of nodes and vgpgpu licences are available – this is managed entirely by SLURM. During Step (2) a SLURM task prolog script will transparently execute. This checks that both VirtualGPGPUPerNode and VirtualGPGPUType are defined. If they are defined, then the prolog script requests a list of backend VMs from the Registry and then sets up the API-Interception execution environment. In Step (3) the GPGPU job will execute as normal; Finally, in Step (4), a SLURM task epilogue is transparently invoked to signal to the Registry that the backend VMs be made available for another job. However, the licence counter will not be incremented until the job exits in Step (5).

Figure 4.3: The flow of a vGPGPU job through the SLURM LRMS



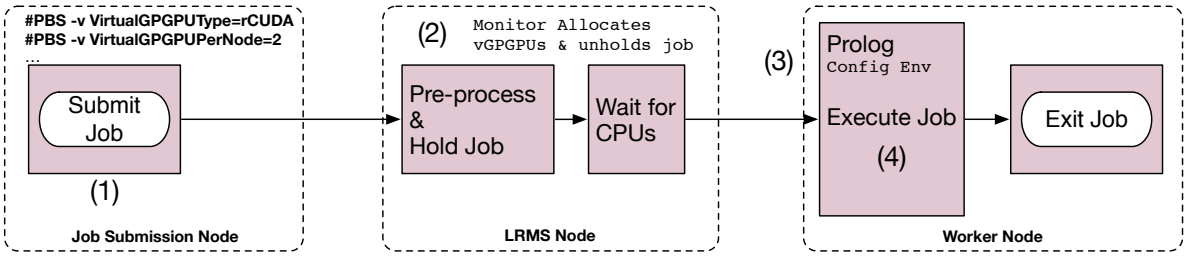
Use-case 2: TORQUE/MAUI

TORQUE/MAUI is a more complex use-case because it has limited support for generic consumable resources (implemented as a MAUI software patch), and limited support for floating consumable resources (p. 16) – it can only decrement a consumed resource by 1 at a time. To bypass these limitations, a new job pre-processing service and a monitoring service have been developed. The flow of a TORQUE/MAUI vGPGPU job is illustrated in Figure 4.4. In Step (1) the number of frontend nodes is defined to be the number of nodes; the VirtualGPGPUPerNode and VirtualGPGPUType are declared as variables, and these will be *exported* to the WN environment. In Step (2) a pre-processing filter examines the job definition; if the VirtualGPGPUPerNode and

VirtualGPGPUType variables are set, an additional job directive is inserted to instruct TORQUE to place the job into a *holding* state; furthermore, the filter injects an additional call to a prolog script. The held job can only be released by an external *Monitor*. Once the job is released, Step (3), the prolog requests a list of backend VMs from the Registry and configures the job environment. Finally, in Step (4) the job is executed and the job exits. The TORQUE/MAUI implementation does not execute an epilogue script – backend VM recovery is left to the Monitor service.

The Monitor service continuously executes on the node where the LRMS runs. It has LRMS operator privileges, allowing it to unhold jobs. During each iteration, the Monitor implements garbage collection of completed vGPGPU job backend VMs and releases them for further use. The Monitor queries the number of free resources, and then iterates over the list of held jobs; if there are sufficient free backend VMs, then the Monitor requests that the required backend VMs are allocated to that job on its behalf, and the job is then released from its hold state. Unheld jobs must wait for available CPU cores. Only one job is released at a time, and this ensures their requests are free of race conditions.

Figure 4.4: The flow of a vGPGPU job through the TORQUE/MAUI LRMS



4.3 Grid-enabled Virtual-GPGPUs

The previous sections developed an abstract model for integrating vGPGPUs into an LRMS (Section 4.1) and demonstrated the application of the approach to some popular LRMSs (Section 4.2). Taking a similar approach, i.e. developing the abstract model first and then developing a concrete implementation, an abstract model for the integration of vGPGPUs into Grid environments is developed below (Figure 4.5) and demonstrated by proof-of-concept.

The most significant differences between the abstract LRMS and Grid models are:

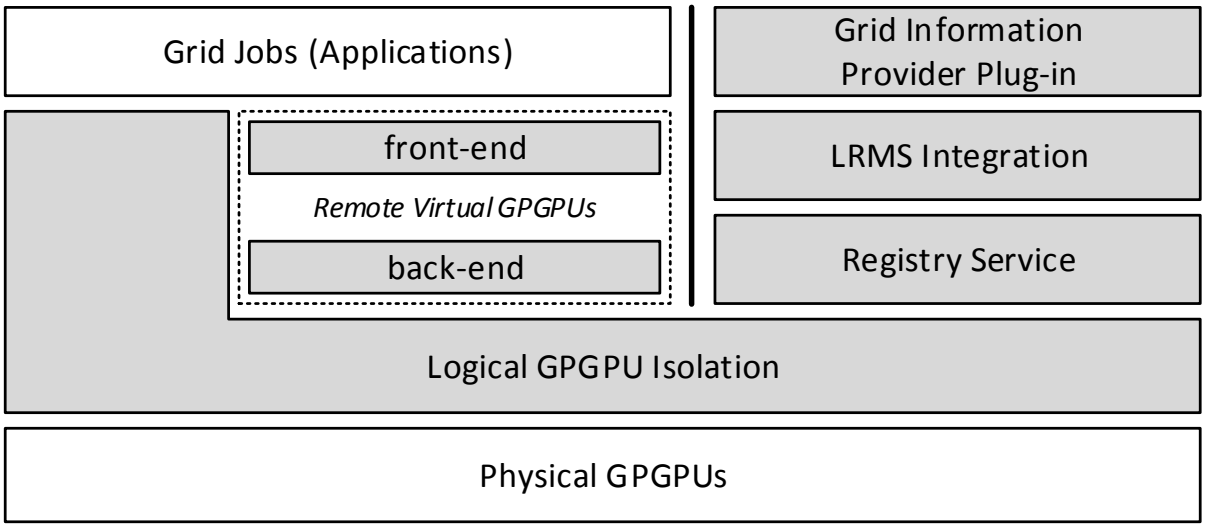


Figure 4.5: Abstract model for the integration of virtual GPGPUs with existing grid infrastructures

(i) how job vGPGPU support is handled by Grids; and

(ii) the need for an additional component to publish the resource status on the Grid.

There are, however, also some subtle differences in other components. The presentation of the abstract grid model will concentrate on those differences.

4.3.1 Virtual CDERs

It was argued in Chapter 3 that the Grid Resource Integration Principles could be used to help integrate CDER resources such as GPGPUs or FPGAs. However, the backend resources managed through the Registry are not CDERs – they are only bound (in this case they are "virtually" attached) to worker nodes at runtime. To resolve this problem, a new definition that accommodates this runtime virtual binding is introduced.

Definition: A *virtual CPU-Dependent Execution Resource* (vCDER) is defined here to be a computational resource that is characterised by the following properties:

(i) access to the resource requires a CPU to be allocated by the LRMS,

(ii) there are a limited number of resources,

(iii) each resource is *virtually* bound to a specific worker node,

(iv) the user that has been allocated the resource perceives they have exclusive access to it,

(v) the resource has its own set of physical characteristics and measurable properties (e.g. memory capacity, performance, hardware characteristics).

4.3.2 Grid Support for vGPGPUs

As the abstract model is derived from the abstract LRMS model, only the differences to the abstract LRMS model shall be considered. In the proposed Grid model, each physical GPGPU is first logically isolated as an independently allocatable resource. This provides for the allocation of a single local GPGPU to a job. Next, API-Interception is used to provide jobs with access to one or more *virtual GPGPUs* or *vGPGPUs*. The vGPGPUs are pooled under the control of a *Registry Service* which, when integrated with a grid Local Resource Management Service (LRMS), allows the vGPGPU resources to be published, discovered and allocated to jobs as required. The model is illustrated in Figure 4.5 and the layers of the model are described below.

Unchanged Components

The abstract model does not require any changes to the Logical GPGPU Isolation or the Remote Virtual GPGPU components.

Grid LRMS Integration

Unlike the pure LRMS case, the user's job originates from a grid User Interface (p. 49), rather than from the cluster's Job Submission Node (p. 16). The grid Job Description Language must be extended to support the abstract requirements previously specified in the LRMS case. When the job is (eventually) submitted to a Compute Element for execution, these requirements must be determined (from a copy of the Job Description) and translated into equivalent LRMS directives.

vGPGPU Registry Service

The only change to the the Registry Service is how the the "query" function is used. In the Grid case, the query function is used to support the reporting of capacity and utilisation information – this will be used by custom Grid Information Provider Plugins to facilitate resource discovery on the grid.

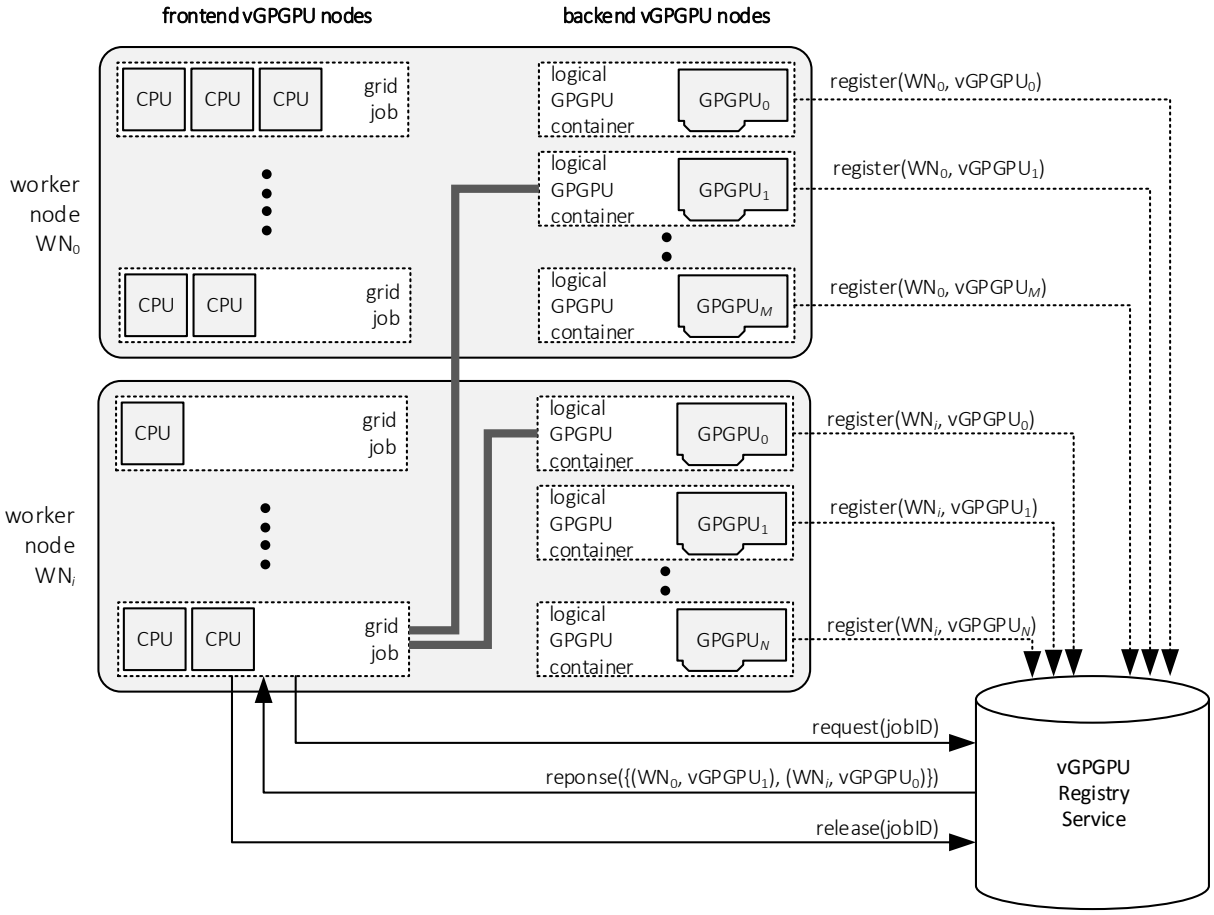


Figure 4.6: The vGPGPU Registry Service. On creation, each vGPGPU backend node registers itself with the vGPGPU Registry Service. The figure illustrates a job executing on worker node i. When the job begins executing, it requests the allocated number of vGPGPUs from the Registry Service. The Registry Service responds with the information required to construct the vGPGPU frontend node, in this case, logically connecting the job to the two allocated vGPGPUs: $vGPGPU_1$ on remote worker node WN_0 and $vGPGPU_0$ on the same local worker node WN_i. (It is important to note that although the GPGPUs are shown in this example to be co-located with the grid worker nodes, this need not be case. It may, however, be desirable to support allocation of local GPGPUs to jobs, where specifically requested.)

Grid Information

Our conceptual model will only specify the type of information published. Namely, the model must accommodate the publication of the type of remote virtualisation supported (e.g, rCUDA and/or VCL), and the number of virtual resources available (both capacity and utilisation). It is assumed that information about the GPGPU hardware itself is published, and that this will also aid resource selection. The information will be published in accordance with the GLUE Schema specification. An API-Interception implementation is presented in Section 4.3.3, demonstrating both how and what information is published. The technique builds upon the work presented in Chapter 3 to publish information pertaining to regular GPGPUs, and therefore publishes details not only about the GPGPU properties, but the relevant vGPGPU information also.

Job Specification, Resource Matching and Submission

GPGPU applications are not uniform in their resource needs. The grid job specification language must be able to satisfy the diverse requirements of single-CPU/single-GPGPU, single-CPU/multi-GPGPU and multi-CPU/multi-GPGPU allocations. The goal of this work is to allow grid users to be able to specify job requirements that (a) allow the user to identify resource centres that support vGPGPUs; (b) identify how many vGPGPUs are available at each resource centre, as well as the characteristics of the vGPGPUs; and (c) how many are allocated and in use. Section 4.3.3 discusses a prototype implementation using API-Interception based vGPGPUs. Here, the two-phase approach used in Chapter 3 that was developed to support regular GPGPUs is expanded upon to enable vGPGPU job specification, discovery and submission through a grid workload management system.

4.3.3 Implementation

It has already been shown that the mechanism for orchestrating the execution of a grid job follows a complex chain of events (Section 2.4.1). As with the GPGPU integration case in Chapter 3, it needs to be restated that integration of new grid resource types, such as GPGPUs or vGPGPUs, is non-trivial, and if these resources are to be provided by *existing* grid infrastructures that are *in-production* and in continuous use by an extensive community of users, the integration challenge is more acute. Adding support for new resources cannot be dependent, for example, on architectural changes,

the replacement of core services or modifications to the GLUE schema. Instead, the approach taken must integrate with existing infrastructure. The approach taken in this prototype is to reuse the modular middleware *hooks* scheme developed for GPGPU integration (Section 3.3), as this was already shown to require relatively small changes to existing grid middleware.

The implementation of vGPGPU into an LRMS (Section 4.3.2) used API-Interception to provide jobs with access to virtualised GPGPUs. The choice of certain API-Interception technology over others depended upon the GPGPU APIs needed by user applications and depended on the current level of support offered by the developers of the API-Interception software (e.g. keeping up with recent releases of CUDA and OpenCL). It is for these reasons that rCUDA and VCL were selected for use.

The prototype infrastructure was developed using the UMD gLite grid middleware and consists of a User Interface (UI), a Workload Management System (WMS), a Top-Level BDII, grid-security infrastructure services, and resource centres using the CREAM CE, along with domain BDIIs. The CREAM CE uses Adaptive Computing's TORQUE 2.5.7 [19] as the batch system server and the MAUI batch scheduler. To enable grid access to the vGPGPUs, the prototype makes three non-intrusive additions to the grid middleware:

(i) a new Grid Information Provider plug-in to publish vGPGPU capacity and utilisation information according to the GLUE 2.0 standard;

(ii) a new layer, added to the standard UI job submission mechanism, which supports vGPGPU resource specification and resource matching using attributes derived from the new vGPGPU GLUE attributes; and

(iii) a modification to the CREAM CE to parse the vGPGPU attributes and pass these as part of the job specification to the batch system. The attributes are used to interact with the Registry Service to reserve and allocate vGPGPUs and to configure the frontend node execution environment in which the job runs.

Logical GPGPU Isolation

Logical GPGPU Isolation is unchanged from the LRMS-case.

Remote Virtual GPGPUs (vGPGPUs)

Again, the same method is used as in LRMS-case.

vGPGPU Registry Service

The vGPGPU Registry Service manages the pool of backend vGPGPU resources. The service implements a simple HTTP-based protocol, which is illustrated in Figure 4.2. Status queries provide the information that is used by Grid Information Provider plug-ins to publish vGPGPU information in GLUE format in the Grid Information System. The registry service is designed to be independent of the LRMS.

LRMS Integration

There are two aspects to LRMS integration: (a) Converting the grid-user's vGPGPU job requirements into corresponding LRMS directives or passing them into the worker node environment for further use and, (b) interacting with the vGPGPU Registry Service.

A typical vGPGPU job specifies the required number of vGPGPUs and the type of vGPGPU Interception API to be used (rCUDA, VCL). As a simple set of environment variable names and an their associated semantics were defined for the LRMS integration case, it is reasonable to reuse these names within the grid job description as a set of keys and values that have a specific interpretation when used with grid job submission. These new key and value pairs are defined in Table 4.1. To pass these into the LRMS and the frontend vGPGPU worker nodes, the method previously used to enable GPGPU job submission on a gLite/UMD-based grid (p. 99) is extended to deal with the new keys and values – a new "hook" is added so that when a grid job is submitted to the CE, the CREAM service inspects the job's JDL for these new vGPGPU-related key/value pairs. If present, these values are either translated into a corresponding LRMS directive, or are passed on to the worker node (as environment variables) where they are subsequently used to help construct the vGPGPU environment and to obtain the required vGPGPUs through the Registry Service.

Grid Information

Information is published about each of the the API-Interception technologies (e.g rCUDA, VCL) supported on the CE. Publication uses the same method described

Table 4.1: Extended JDL specification processed by the LRMS

Job Specification Key	Value and Meaning
VirtualGPGPUPerNode	Specifies the number of vGPGPUs required (a positive integer)
VirtualGPGPUMethod	A text string specifying which API-Interception is used (e.g. "rCUDA" or "VCL")

in Chapter 3 (p.95). However, rather than publishing the details as GLUE 2.0 ExecutionEnvironments, the VCL and rCUDA software can be published as GLUE 2.0 *ApplicationEnvironment* entities, and extended using the Attribute-Extension strategy. Each entity publishes additional information regarding the number of installed vGPGPUs available and the number of free vGPGPUs (Table 4.2) available under rCUDA and VCL. Information about the hardware is published in the ExecutionEnvironment, in a manner that is similar to the GPGPU integration work presented in Chapter 3.

Table 4.2: GLUE 2.0 Entities

Application Environment	Attribute	Description
rCUDA		rCUDA vGPGPU supported on CE.
-	FreeVGPGPU	Count of available rCUDA-enabled vGPGPUs.
-	TotalVGPGPU	Total installed rCUDA vGPGPUs.
VCL		VCL vGPGPUs supported on CE.
-	FreeVGPGPU	Count of available VCL-enabled vGPGPUs.
-	TotalVGPGPU	Total installed VCL vGPGPUs.

Job Specification, Resource Discovery and Submission

The *two-phase* solution (Section 3.3.5) is updated to extract attributes from both the ExecutionEnvironment and ApplicationEnvironment during the first phase. As the "GPGPUPerNode" is no longer relevant to the vGPGPU environment, this key and value pair is dropped during the creation of the ClassAd Machine Resource Offer, and a new expression using the "TotalVGPGPU" value from the relevant (rCUDA/VCL) ApplicationEnvironment is applied instead. The ClassAd Resource Request will also use the Requirements sub-expression:

$$VirtualGPGPUPerNode \times CPUNumber <= TotalVGPGPU_{VirtualGPGPUMethod}$$

In this way, only Compute Elements satisfying that sub-expression are considered.

4.3.4 The Grid Resource Integration Principles

The development of the Grid Resource Integration Principles was central to the integration of GPGPU-based CDER resources in Chapter 3. Certainly, the vGPGPU Grid integration satisfies the Discovery principle, as information about the GPGPUs and the API-Interception software are published. Furthermore, Independence was guaranteed by construction – indeed, the choice of resource names were arbitrary so it was possible to maintain the same naming convention for handling the resources across a range of LRMSs and across the grid Job Description Language. Finally, Exclusivity was also guaranteed by construction. Each GPGPU was isolated in a virtual machine, and the allocation of these resources through the Registry ensured that they were uniquely assigned to the user.

4.4 Evaluation

To evaluate the performance of the proposed virtual GPGPU architecture, a number of experiments were conducted. The first experiment investigated the impact of using multi-layered GPGPU virtualisation. The second experiment looked at the throughput of vGPGPU jobs in comparison to regular GPGPU jobs with similar requirements, aimed at determining how well the vGPGPU-enabled Compute Element (or LRMS) could process vGPGPU jobs.

4.4.1 The Performance of Multi-Layered Virtualisation

The first set of experiments is to determine the impact that each virtualisation layer (Docker for OS-Level GPGPU isolation and rCUDA/VCL for API-Interception) has on the performance and viability of the prototype. Two simple experiments were selected. The first examines the performance of a compute-intensive GPGPU application with minimal communication, while the second is a bandwidth intensive application that moves data from the worker node to the GPGPU and back. Five scenarios were studied to help establish how each layer impacts performance. Although there are existing studies analysing the raw performance of GPGPUs [57] [52], API-Interception with rCUDA [62], and OS-virtualisation [71], there are no studies analysing the cumulative effects of multi-layered virtualisation. The five scenarios are:

(i) Native performance with direct access to the GPGPU (Local);

Table 4.3: Execution data for Nvidia Black-Scholes application (1,000 invocations)

Access Method	Total Time (1000 jobs)	Relative Performance	Average per-job GPUTime (msec)	Average per-job Memory Bandwidth
Local	2773.52s	1.0	3.59	22.26 GB/s
Local-rCUDA	2810.99s	1.014	3.60	22.25 GB/s
Local-rCUDA/Docker	2831.12s	1.021	3.60	22.24 GB/s
Remote-rCUDA	3524.73s	1.271	3.60	22.24 GB/s
Remote-rCUDA/Docker	3524.69s	1.271	3.60	22.25 GB/s

(ii) worker node access with rCUDA running locally on the worker node (Local-rCUDA);

(iii) worker node access to a local GPGPU through rCUDA and a Docker container (Local-rCUDA/Docker);

(iv) worker node access to remote GPGPU using rCUDA only (Remote-rCUDA); and

(v) worker node access to a remote GPGPU using rCUDA and Docker (Remote-rCUDA/Docker).

The network fabric uses 1Gbps Cat5e Ethernet. The GPGPUs were Nvidia GTS 450s. The compute intensive application was executed 1,000 times for each scenario, while the bandwidth intensive application was executed 100 times.

Experiment 1:

The Black-Scholes application is provided by Nvidia to demonstrate how GPGPUs can be used to calculate the price of European financial market options. This application is distributed with the Nvidia CUDA Software Development Kit. Input values and initial conditions are hard-coded into the application, and a total of 8,000,000 options are calculated. There is minimal data transfer between the CPU and the GPGPU, so this application is a good indicator of how well an application will perform when it is not dependent on network I/O. The results in Table 4.3 show the total time taken to run each GPGPU scenario. The ratio between the time taken to run and the corresponding time taken on the local GPGPU is shown. The results consistently indicate that in both Local and Remote GPGPU cases, the combination of rCUDA and Docker has a negligible impact on the overall runtime in comparison to just using rCUDA alone.

Experiment 2:

The BandwidthTest application also comes from the Nvidia Software Development

Table 4.4: Execution data for Nvidia BandwidthTest application (100 invocations)

Access Method	Total Time (100 jobs)	Relative Performance	Host to Device Bandwidth	Device to Host Bandwidth
Local	55.10s	1.0	3224.27 MB/s	3268.48 MB/s
Local-rCUDA	72.96s	1.32	2924.77 MB/s	1701.18 MB/s
Local-rCUDA/Docker	92.90s	1.69	1227.15 MB/s	1632.54 MB/s
Remote-rCUDA	665.05s	12.07	114.89 MB/s	120.61 MB/s
Remote-rCUDA/Docker	665.10s	12.07	114.87 MB/s	122.52 MB/s

Kit. Its purpose is to measure the memcopy bandwidth of the GPGPU and memcpy (memory copy) bandwidth across the PCI-e bus. In the case of rCUDA and Docker, the application should generate significant network I/O that will have an impact on its performance. The results for these application runs are tabulated in Table 4.4.

The results show that even locally, rCUDA and Docker will have a noticeable impact on the application performance. The performance of remote GPGPUs is very poor over 1Gbps Cat5e Ethernet, with the bandwidth test taking over twelve times longer than running the same application locally.

4.4.2 Batch Workload Performance using the LRMS

A small-scale experimental deployment of the virtual GPGPU implementation described in Section 4.3.3 has been used to evaluate the expected performance of the vGPGPU model. Naturally, when using remote vGPGPUs, the execution time for a single job will be expected to be greater than the execution time achieved using local GPGPUs, due to the communication overhead that is introduced. The scale of this slowdown was illustrated in Section 4.4.1. From a performance perspective, the main goal of the vGPGPU model is to reduce the overall execution time for a workload consisting of many jobs, rather than reducing the execution time of any individual job. The aim is to achieve this through increased utilisation of grid Compute Element or LRMS resources (both GPGPUs and CPUs in this case), leading to shorter waiting times for jobs. In particular, the vGPGPU model is expected to provide opportunities to execute waiting jobs (using remote GPGPUs) which would not otherwise exist. The performance for a single individual job is sacrificed to improve the overall performance for the workload.

The experimental vGPGPU deployment consisted of five worker nodes, each with 8 CPU cores and 2 Nvidia GTS 450 GPUs. The worker nodes were interconnected using a Cat5e Ethernet-based network through a single switch. The nominal data rate

was 1 Gbps. The performance of the simple deployment providing direct access to physical GPGPUs was compared with that of the vGPGPU deployment. Two different vGPGPU allocation strategies were implemented: (i) the worker nodes are assigned randomly allocated vGPGPUs (random); and (ii) where possible, the worker nodes are assigned available Local-vGPGPUs (prioritised).

Workloads consisting of 100 jobs with varying relative proportions or "mixes" of one-GPGPU and two-GPGPU jobs were applied to both the physical GPGPU (pGPGPU) deployment and the experimental vGPGPU deployments (random and prioritised). In the results presented below, *Mix_X_Y* represents a workload containing X one-GPGPU jobs and Y two-GPGPU jobs. Each job executed 20 invocations of the GROMACS [157] 5.04 molecular dynamics application serially. Sample configuration and data came from the GROMACS *angle1* regression test. The parameters were as per the original sample configuration, with the exception of using the *Verlet* cut-off scheme (required for GPGPU execution), a step-size of 1,000, and using two CPU threads. All of the jobs were submitted directly to the LRMS (i.e not using the Grid) to ensure fast job-submission rates. Each Mix was executed with 5 samples. The three different job throughput scenarios were investigated and evaluated. The total number of sample jobs submitted for all physical GPGPU (pGPGPU) and vGPGPU workloads was 7,500, with the GROMACS application executed a total of 150,000 times.

Figure 4.7 and Table 4.5 show the average aggregate workload completion times (in seconds) for a variety of workloads using a mixture of one- and two-GPGPU jobs, as well as the the total time to execute the complete set of workloads under pGPGPU and vGPGPU (random and prioritised) allocation policies. The aggregate values includes the time that tasks may wait before being executed. It can be seen from Table 4.5 that the overall throughput of the vGPGPU using random allocation (7h 10m) approaches that of the pGPGPU deployment (7h 01m), however prioritised vGPGPU allocation executes some 20 minutes quicker (6h 41m) than pGPGPU allocation, despite the extensive use of slower remote GPGPUs by jobs. This indicates that the vGPGPUs can be allocated to jobs more effectively than the pGPGPU case.

A breakdown of the experimental data in Table 4.6, Table 4.7 and Table 4.8 summarises the observed execution times of identical workload mixes in each scenaio. Furthermore, Table 4.9 shows that moving from using one local physical GPGPU to a vGPGPU on the same host increases the runtime by approximately 11%-12%. Similarly, the cost of executing the application on two vGPGPUs is dependent on the

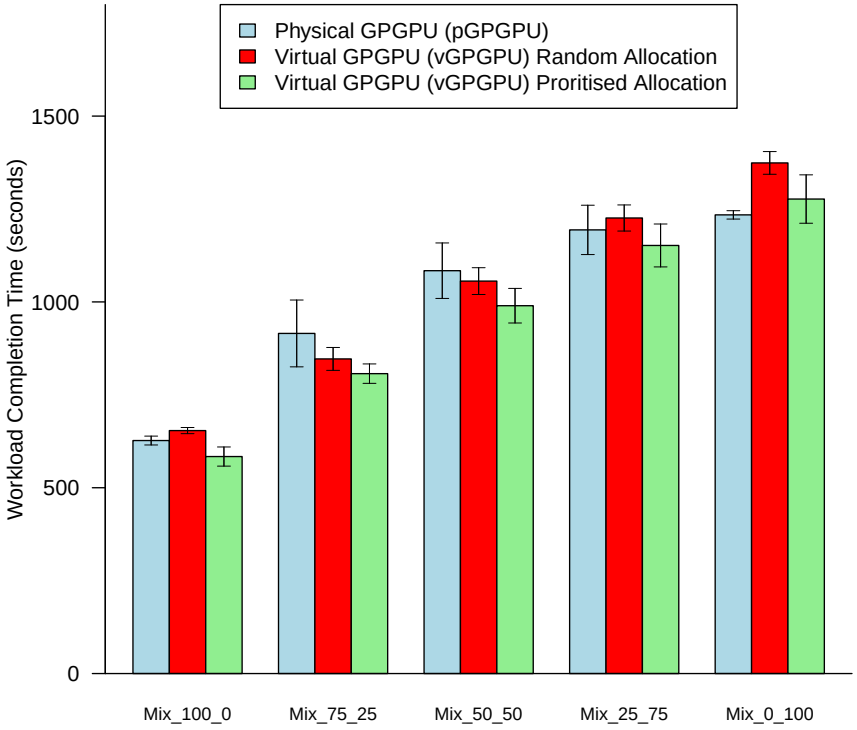


Figure 4.7: Average workload completion time (including waiting time) for a vGPGPU and pGPGPU deployment and for workloads with varying relative proportions of one-GPGPU, two-GPGPU jobs.

Table 4.5: Average workload completion times for vGPGPU and pGPGPU workload mixes

Workload	Average Runtime pGPGPU	Average Runtime vGPGPU (random)	Average Runtime vGPGPU (prioritised)
Mix 100 0	627.0 sec	653.8 sec	583.9 sec
Mix 75 25	915.2 sec	846.6 sec	811.2 sec
Mix 50 50	1084.0 sec	1056.2 sec	993.7 sec
Mix 25 75	1193.8 sec	1225.8 sec	1151.9 sec
Mix 0 100	1234.2 sec	1374.0 sec	1276.6 sec
Total Time	7h 01m	7h 10m	6h 41m

number of "local" GPGPUs assigned to each job and ranges from approximately 20% when local to 42% when both vGPGPUs are remote. The differences in these costs can be attributed to the extra cost of communication due to how Docker manages container networking, as well as the physical Cat5e Ethernet-based network.

Table 4.6: Average runtime (seconds) of 20 invocations of GROMACS angle1 regression test with step-size=1000, 2 CPU threads and Verlet Cutoff Scheme (pGPGPU)

Workload	1-GPU	Std. Dev 1	2-GPU	Std.Dev 2
Mix 100 0	45.62	0.76	N/A	N/A
Mix 75 25	45.08	0.84	46.43	0.77
Mix 50 50	45.01	0.90	46.31	0.90
Mix 25 75	44.78	0.76	46.26	0.88
Mix 0 100	N/A	N/A	46.26	0.94

Table 4.7: Average runtime (seconds) of 20 invocations of GROMACS angle1 regression test with step-size=1000, 2 CPU threads and Verlet Cutoff Scheme (vGPGPU)

Workload	1-GPU	Std. Dev 1	2-GPU	Std. Dev 2
Mix 100 0	57.20	3.82	N/A	N/A
Mix 75 25	56.78	4.79	63.54	3.86
Mix 50 50	58.60	4.24	64.51	4.56
Mix 25 75	56.90	5.07	63.29	4.67
Mix 0 100	N/A	N/A	61.72	4.31

Table 4.8: Average runtime (seconds) of 20 invocations of GROMACS angle1 regression test with step-size=1000, 2 CPU threads and Verlet Cutoff Scheme (vGPGPU) with prioritised vGPGPU allocation

Workload	1-GPU	Std. Dev 1	2-GPU	Std. Dev 2
Mix 100 0	51.81	2.57	N/A	N/A
Mix 75 25	53.63	3.75	58.70	3.38
Mix 50 50	54.53	4.66	60.69	5.29
Mix 25 75	53.55	4.55	58.09	4.10
Mix 0 100	N/A	N/A	57.97	4.94

Table 4.9: Average Cost of Docker/rCUDA Virtualisation for 20 invocations of GROMACS angle1 regression test with step-size=1000, 2 CPU threads and Verlet Cutoff Scheme

	1-GPGPU Average	Cost (%)	2-GPGPU Average	Cost (%)
pGPGPU (1 GPGPU)	45.29			
vGPGPU (1 Local-rCUDA/Docker)	50.67	11.86 %		
vGPGPU (1 Remote-rCUDA/Docker)	60.40	33.37 %		
pGPGPU (2 GPGPU)			46.30	
vGPGPU (2 Local-rCUDA/Docker)			55.16	19.16%
vGPGPU (1 Local, 1 Remote-rCUDA/Docker)			60.03	29.70%
vGPGPU (0 Local, 2 Remote-rCUDA/Docker)			65.98	42.54%

It should be noted that in practice, the use of a higher speed and lower latency node interconnection network would be expected to greatly improve the performance

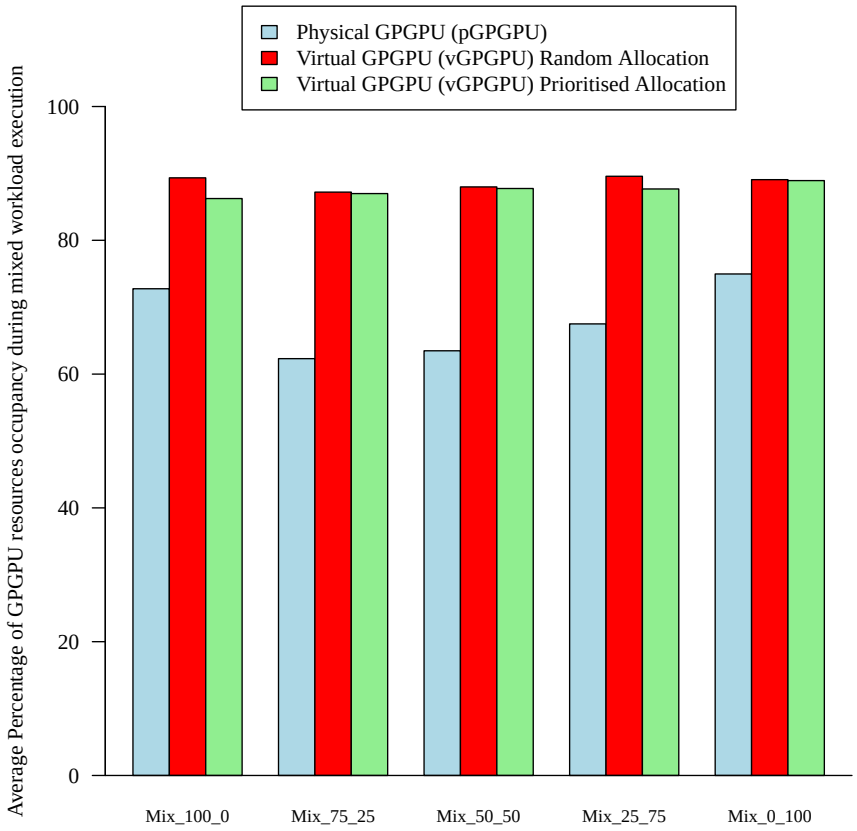


Figure 4.8: Average GPGPU utilisation for a vGPGPU and pGPGPU deployment and for workloads with varying relative proportions of one- and two-GPGPU jobs.

of remote GPGPU access and, based on the results presented here, it would be expected to result in the performance of the random vGPGPU deployment exceeding that of the pGPGPU deployment. It has been shown elsewhere [158] that, using an interconnect such as Infiniband QDR, a reduction in this overhead to 0.67% to 7% might be expected. Indeed, to test the basis of this conjecture, a small-scale experiment examining the use of 10Gbps single vGPGPU GROMAC jobs under a Remote-rCUDA/Docker setup yielded an average runtime of 51.3 seconds, i.e it performed almost as well as the 1 Gbps Local-rCUDA/Docker case.

Figure 4.8 shows the average GPGPU resource utilisation for the same workload executing on both vGPGPU and pGPGPU deployments. The utilisation is measured as the ratio of the total time GPGPUs are allocated to all the workload mix jobs to the total number of seconds that the pool of GPGPUs were available, and includes the workload waiting times. The utilisation function is discussed in detail in Appendix C. The utilisation values plotted in Figure 4.8 are derived from the data in Table 4.5, Table 4.6 and Table 4.7.

Clearly the vGPGPU deployment achieves better resource utilisation. This further supports the contention that any improvement in network performance or GPGPU scheduling will cause the performance of the vGPGPU deployment to exceed that of the pGPGPU deployment.

4.5 Other Related Work

The closest related work that integrates vGPGPUs into grids are GridCuda [63] and WNDoES [72]. The former uses a registry to match jobs to vGPGPU resources, however, it is tied to a specific (obsolete) grid technology and a specific type of resource (Nvidia GPGPUs). The latter works on EGI's UMD middleware, but does not implement any resource publication or discovery or job specification. This means the WNDoES solution does not have the flexibility to discover specific details about the state of the GPGPU resources or their capacity and utilisation. The work presented in this thesis strives to be grid middleware (UMD, Globus) and LRMS neutral, and to be flexible in handling many different types of new resources.

Since May of 2015, a European Grid Initiative project (EGI-Engage) has been working on integrating GPGPU resources into grid infrastructures and federated clouds (using virtual GPGPUs). However, the proposed approach is to generate a new GLUE 2.X draft specification, and to invest very significant effort into modifying the Workload Management System. This is a time consuming process – the work presented in this thesis was designed to enable rapid integration and to avoid singular technology specific solutions. Furthermore, the EGI work-plan only uses high-end Nvidia GPGPUs to overcome problems with GPGPU integration into the TORQUE LRMS. The exclusion of good lower-cost GPGPU models is, in this author's opinion, a barrier to the deployment of accelerators by some grid resource providers. In the case of GPGPU virtualisation on the federated EGI Cloud, once again the target implementation is expensive high-end Nvidia GPGPUs that support PCI-Passthrough to virtualise the GPGPUs, and does not consider more general solutions.

The rCUDA developers are working with the SLURM developers to integrate vGPGPU directly into that LRMS. Although this work was mooted to be available in the August 2015 release of SLURM, it has not yet materialised in the public domain. This singular API-Interception solution also suffers from an inability to operate on the same cluster with other API-Interception solutions for reasons explained on Page 121.

4.6 Summary and Conclusions

This Chapter looked at expanding upon the work presented in Chapter 3 to enable virtual GPGPU resources on grid infrastructures. These new resources are now accessible with grid discovery and job specification support. Virtual GPGPUs allow user jobs to access non-local GPGPU resources from any worker node as if they were local, and consequently allows worker nodes (and jobs) to access resources that were previously unavailable due to physical constraints.

The particular approach was to design an abstract model that: (i) proposed a "frontend/backend" architecture that isolated GPGPUs into a pool of "backend" resources; (ii) delegated management of these to an new external service called the Registry; and (iii) proposed that the "frontend" machines could request backend resources from the Registry, thereby allowing the frontend to gain control over the requested GPGPUs. The abstract model also defined how the "frontend/backend" vGPGPU architecture could be integrated into a Local Resource Management System.

Based on the abstract model design, it was shown how to realise an implementation by combining OS-Level virtualisation of the GPGPU with API-Interception. This implementation included a new "Factory" service to construct the backend resources and to aid with their initial registration in the Registry. It was further shown how to apply this realisation to two distinct LRMSs – SLURM and TORQUE/MAUI. In the latter LRMS case, it was necessary to develop a further service called the "Monitor" to keep track of (and assist) vGPGPU jobs submitted through the LRMS.

A second phase to the work in this chapter explored the extension of the abstract LRMS model to an abstract Grid model. Starting with the LRMS realisation, an implementation for Grid was also realised. This realisation amended the CDER Grid Resource Integration Principles developed in Chapter 3. The amendment defined a new virtual CDER classification as the vGPGPUs were not "physically" bound to the worker nodes, but were instead dynamically assigned during the job's execution. The extension of the model to Grid included support for resource discovery, job specification and exclusive access to the vGPGPUs.

An evaluation of the performance of the LRMS/Grid implementation was undertaken. The first experiment looked at the impact that multiple layers of virtualisation had on GPGPU performance. The data from the first set of experiments (Table 4.3 and Table 4.4) showed that: (i) when local (i.e. contained to the same physical hardware)

vGPGPU jobs execute computationally bound applications exhibiting little communication, neither rCUDA nor Docker have a significant impact on performance (1% to 2%). However, there was a performance impact of circa 25% (Table 4.3) in both remote cases (where the frontend node is on separate hardware to the backend VM); and (ii) the bandwidth experiment results (Table 4.4) show that the performance degradation due to rCUDA and Docker is compounded at a local level, but Docker's impact is masked in the remote case. Both results imply that the Cat5e network infrastructure's 1Gbps bandwidth was insufficient, and that further tests, such as using 10Gbps (p. 141), were needed to see if any improvements can be made to the TCP/IP performance under both rCUDA and Docker.

A second set of experiments compared the throughput of a set of jobs through a vGPGPU-enabled LRMS to the same set of jobs executing with physical GPGPUs. Despite being restricted to using a relatively low-performance network interconnection between worker nodes and their vGPGPUs, the throughput performance results show that a vGPGPU allocation policy that prioritises the allocation of vGPGPUs (based on whether the vGPGPU is local to the worker node) performs better than the corresponding physical GPGPU throughput, and even an unoptimised (random) allocation of vGPGPUs to worker nodes comes close to achieving the same performance as regular GPGPUs. Again, the test of using 10Gps network interconnects (p. 141) has shown that the application executed in circa 51.30 seconds (p. 141), a time that is comparable to Local-rCUDA/Docker execution times (Table 4.9) It is hypothesised that further evaluation using higher performance network interconnection technologies and improved GPGPU scheduling is expected to yield further increases in performance, whose throughput will out perform the throughput of a non-vGPGPU system.

Chapter 5

Conclusions and Future Prospects

5.1 Summary and Contributions

Grid infrastructures, which have been central to some of the largest computationally intensive scientific investigations in recent times, now need to be able to rapidly adapt to changes in the provision of new grid services – in particular, those new services that provide access to massively parallel computing facilities. As the complexity of computational simulations and models increases, the dependency on faster processing power and massively parallel computing also increases. In synergy, faster processing power drives advancements in computational simulations and models. The emergence of several new types of computational resources, such as GPGPUs, Intel's Xeon Phi, FPGAs and other types of accelerators are a manifestation of this. These resources have diverse characteristics (e.g. physical properties, supporting software, access mechanisms), so there is no uniform way to discover them on the Grid, nor to specify them in a grid job – this is a resource integration problem.

The resource integration problem is not just caused by the diversity of the resources themselves, but also by other factors such as the diversity of the systems (e.g. hardware, software, operating systems and resource managers) that make up Grids. The latter is managed by adherence to common standards that define how systems, services and users interact with Grids. While this helps provide for long-term stability, the downside is that Grids cannot handle rapid changes to the standards and grid middleware (grid software). The *Resource Integration Problem* can be summed up as a question of how to deal with the integration of new computational resources in a massive distributed grid computing environment that (somewhat conversly) requires stability.

The hypothesis of this thesis is that a conceptual approach can be used to provide a level of abstraction that assists with the process of integrating new computational resources that adhere to existing grid standards and require only a small number of unobtrusive changes to the grid middleware. This has never been explored before. The objective was to investigate and develop a flexible, dynamic approach to the integration of new (and yet to be conceived) resources into existing grid infrastructures. This has been achieved. The philosophy, abstractions and principles employed have yielded a hard-won demonstrated proof of principle.

The Conceptual Approach to Resource Integration

The philosophy constrained solutions to minimal changes to the existing infrastructure, middleware or standards as a precondition for rapid integration of new resources.

The abstractions proposed a classification for resources that offer accelerated computing capabilities and are managed through an LRMS (the CDER). CDERs are a subset of LRMS consumable resources, defined so as to encapsulate the fundamental computational role that the resource plays in the execution of an application and to highlight the tight bond between the CPU allocation process and access to accelerator resources.

The principles (Grid Resource Integration Principles) establish a set of high-level requirements that facilitate the integration of CDERs so that they are seen as a constituent part of the grid ecosystem, so the CDER can be discovered and accessed by the grid user in a manner that is similar to discovering and accessing other grid resources such as CPUs. There are five principles: (i) Discovery; (ii) Independence; (iii) Exclusivity; (iv) Accounting; and (v) Monitoring. Only Discovery, Independence and Exclusivity have any role in the execution of a grid application and were therefore the focus of the thesis.

Discovery required at most minor changes to the grid middleware. Similarly, Independence needed only minor changes, as it could be implemented externally via a two-phase resource discovery and selection process layered on top of the existing grid middleware. Exclusivity was managed by introducing a new service that operated at the LRMS level, did not require modification to the grid middleware, and ensured there were no resource over-commitments.

Extensions for CDER Virtualisation

Virtual resources have the advantage of not being physically bound to a machine, and may be used to improve resource utilisation or to build systems that are difficult to construct using just physical resources. The CDER definition was extended to eliminate the physical bond between the worker node and what became the virtual-CDER. No changes to the Grid Resource Integration Principles were considered necessary.

A new abstract multi-layer frontend/backend model for virtual-GPGPU LRMS integration allowed different combinations of virtualisation methods to be used in conjunction with one another, giving it the flexibility to isolate individual resources (backends), and subsequently recombine them within a worker node (frontend) environment. New VM Factory and Registry services were developed to facilitate the creation and management of these new resources. A third service provided an LRMS specific interface to manage resource allocation requests. Extension to the Grid followed the same pattern as GPGPU integration.

Contributions

The key contribution of this thesis is the development of the Conceptual Approach and its Grid Resource Integration Principles. The Conceptual Approach provides a dynamic and flexible mechanism to assist with the integration of new types of resources, physical and virtual CDERs, into the Grid. This enables their discovery, selection and job submission on the Grid, which had been difficult to implement in stable grid environments. It was shown that by using the Conceptual Approach only a few minor changes to the grid middleware were required, and the tools and services that helped provide this extra functionality exist outside the grid-layer. Other key contributions of the thesis are summarised as follows:

(i) The development of the conceptual strategies and the subsequent investigation of published CDER information resulted in a detailed analysis of the advantages and disadvantages of each strategy. The conclusion of this analysis was that the strategies that used GLUE 1.X were not capable of dealing with the complexity of new accelerator resources nor their associated dynamic data. Only strategies using GLUE 2.0 or above could be recommended. The Attribute-Extension strategy had the advantage that it was easier to pack data in a few GLUE entities,

resulting in a smaller data footprint. Furthermore, collating, parsing and processing that data was also much easier than reassembling them from many different Class-Extension entities (the alternative).

(ii) The development of a lightweight GPGPU resource management service that provides capabilities not currently supported in some LRMSs (e.g. the widely used TORQUE/MAUI resource manager). The subsequent application of the Grid Resource Integration Principles resulted in the full integration of GPGPU resources into the Grid with GPGPU discovery, selection and job submission facilities. This was supported by using a new two-phase job submission system. A number of additional services and applications were developed as part of this integration effort.

(iii) The design and development of an abstract frontend/backend model that was the basis for the integration of virtual-GPGPUs into both cluster and grid environments.

(iv) The implementation of a prototype Grid supporting virtual-GPGPU resources that adheres to the Grid Resource Integration Principles. This prototype introduces a new multi-component system that supports multiple API-Interception implementations on several LRMSs. The system consists of: (a) a Factory component that produces lightweight Linux Containers, where each Container supports one or more API-Interception implementations and controls a single GPGPU; (b) a Registry service that manages the Container resources; and, (c) a set of plug-in scripts that bridge between the LRMS and the Registry. The key investigation into the performance of this system showed that, in spite of slower network connections, it was possible to efficiently allocate virtual-GPGPUs such that the throughput of jobs on virtual-GPGPUs almost matched similar jobs on physical GPGPUs. There was scope to improve performance with higher bandwidth networking. Furthermore, it was possible to provision access to more virtual-GPGPUs from a single worker node, than is currently possible with the normal physical GPGPU model.

Software Reuse

There are potential contributions that extend to domains outside the field of grid computing, namely:

(i) The development of the two resource management services, the worker node GPGPU allocation service for TORQUE/MAUI and the Registry Service for managing virtual-GPGPUs, have potential re-application outside of the target grid environment, for example in the wider Cluster Computing environment. More generally, it is envisaged that the two registry systems can be developed further to assist with the integration of other resources, especially those which are are not currently accommodated for in some LRMSs. For example, this could include managing the allocation of multiple FPGAs on a single worker node.

(ii) The VM Factory, together with the Registry, has application outside of the LRMS environment. For example, the VM Factory can be expanded to search for other types of physical hardware devices and to construct unique virtual machines that provide isolated environments for those devices. The VM registration protocol and Registry can be adapted to help maintain a VM/device inventory.

5.2 Limitations and Alternatives

Although the thesis and research publications describe several experiments, it is important to identify where further research opportunities were available and had been considered, but the opportunity to pursue them did not present itself. In addition, the limitations of the GPGPU and virtual GPGPU prototypes merit discussion, as well as what further development and refinement may have been possible.

Publication and Handling CDER data

In Chapter 3 two promising strategies for handling CDERs were identified. The first, Attribute-Extension, packs all the associated CDER data into a single GLUE entity instance, while the second strategy, the Class-Extension, creates a set of new GLUE Extensions instances that link back to an associated entity by way of a Foreign Key (p. 87).

The two-phased job submission solution was implemented using the Attribute-Extension method because: (i) the solution's published data footprint was significantly smaller than the Class-Extension strategy (Table 3.3); and (ii) Attribute-Extension had the advantage of facilitating simpler publication of the GLUE data and to develop parsers that convert the published CDER data into ClassAd Resource Offers (which helped identify suitable resource providers). Furthermore, as the solution is simple, it is consistent with the objectives of the thesis – integration of CDERs must not require radical changes to the grid middleware. However, there are alternative solutions, that were not explored in the thesis:

Class-Extension This would require developing one or more grid information providers to create new GLUE instances for each individual attribute (and thereby greatly increasing the total amount of data that needs to be processed by the Grid Information System). Furthermore, each instance must be linked back to a standard GLUE entity (e.g. an ExecutionEnvironment or ApplicationEnvironment) by way of a Foreign Key. To create an equivalent ClassAd offer under this strategy requires finding all Extension instances matching the CDER attributes, partitioning the Extension data using the Foreign Keys, and then generating the ClassAd Machine offers.

Hybrid This approach, using both Attribute- and Class-Extension strategies to describe the CDER attributes and states, would be a more complex, but potentially more efficient, solution. For example, the hybrid approach could use Attribute-Extension for describing static properties, whereas Class-Extensions could be used for dynamically changing data. Similar to the Class-Extension approach, the complexity lies in gathering the CDER data and converting it into a ClassAd Resource Offer. This hybrid solution would be a compromise on space (publication) versus time (lookup and conversion to ClassAd) – in particular, if there are relatively few dynamic attributes the space overhead may be small, whereas time efficiencies could be gained by searching for particular (dynamic) Key/Value pairs as there is less complexity in parsing this data and converting it to ClassAd expressions. The hybrid approach needs to be explored in future work.

CDER Namespaces

The naming conventions used to advertise the CDER resource attributes in the thesis are arbitrary, but consistently named (e.g. GPGPUClockSpeed or VirtualGPGPUPerNode). One of the lessons learned from the MPI/Grid integration effort was that the users and resource providers needed to agree on a common naming scheme that described the MPI service, and have a common semantics for describing and publishing the resource provider specific software or hardware setup. This allowed the users to discover the resources and to exploit them in a consistent manner across the Grid. One potential technical solution that could be used to alleviate the issue of CDERs being named the same, but used differently (i.e. namespace collision) is to develop a Central Registry of CDERs. This Registry would allow new CDERs to be recorded. The registration process would require that the CDER is well defined with a description (schema) of its use and attributes (Keys, Values and their semantics). A unique identifier (e.g a UUID) could be generated for each new CDER schema definition to ensure there is no collision in how the CDER is advertised.

The Central Registry could be a searchable web-service that allows both users and resource providers to search for existing CDER definitions (perhaps using keywords such as "GPU"). The Registry could be used as a repository for software that provides all the necessary integration hooks into the grid middleware (e.g. grid information providers, LRMS scripts). Resource providers could advertise the unique identifier. The two-phase job submission could use this unique identifier as a resource selection filter and to ensure that the CDER resource requirements are specified according to the registered schema.

The addition of a new CDER resource type and its attribute definition should be managed in consultation with the Grid stakeholders, i.e. the user communities and the resource providers, who would decide on what properties should be defined in the CDER schema, and whether it is mandatory or non-mandatory to publish particular attribute Key/Value pairs.

Enhancing LRMS CDER Management

The implementation of the worker node GPGPU resource management micro-service in Chapter 3 was focussed on overcoming a major resource management deficiency – the handling of Nvidia GPGPU resources under the Torque/MAUI LRMS. This particular

combination of accelerator and LRMS is expected to be the most common one used on the Grid. The core of the solution was developed to help the integration of the CDER into the TORQUE LRMS (non-grid) – further layers that added grid discovery and grid job support help implement a Grid solution. While the development of this micro-service enhances the functionality of TORQUE/MAUI, there is great scope to further develop the micro-service for other uses in the following ways:

(i) expand the range of CDER resources (e.g ATI GPGPUs, FPGAs) that can be handled;

(ii) adopt the micro-service for use with other LRMSs.

In the former case, it is envisioned that the management system would need to modify the service with the inclusion of new database tables (perhaps one table per CDER type). Furthermore, the CDER request/release protocol would need to be expanded to handle each new CDER type.

Enhancing LRMS CDER protection

There are opportunities to further develop the security offered to individual CDER resources on multi-user systems and (non-shared) virtual machines, especially in the area of enforcing CDER access and control at the operating system level. One particular route would be through the development of Linux Control Groups policies (Cgroups) [159] [160]. The use of Cgroups with NVIDIA GPGPU CDERs was investigated during the production of this thesis (by defining and applying a Cgroup policy definition to a GPGPU-enabled container), but issues with interaction of the NVIDIA devices and Cgroups caused system instabilities. The Nvidia system and the SLURM LRMS developers were aware of this issue [161] and had noted that this would be addressed in a future release of the Nvidia device drivers. The update was released in March 2015 [162].

Masking Two-Phase Job Submission

The Two-Phase process developed in Chapter 3 and used in both Chapter 3 and Chapter 4 uses one JDL to specify the CDER requirements, which acts as a filtering process to match CDERs to suitable resource providers, and a second JDL which describes the user's application and non-CDER job requirements. Ideally the CDER requirements

would be merged into a single job description file, where an updated job submission command would mask the CDER resource provider selection step. It should be noted that a single job submission command was developed for the prototypes, but this still required both the CDER job description and the grid job description files.

5.3 Further Evaluation

The early withdrawal of Ireland's National Grid Initiative, Grid-Ireland, from participation in pan-European collaborative grid infrastructures and projects in January 2013 made it impossible to extend the testing of the prototype beyond a local testbed. Development and testing was only made possible by building a local grid infrastructure with a private certificate authority, virtual organisation management systems, and other services that were essential to grid infrastructures. Ideally, the GPGPU and vGPGPU prototypes should have been tested on a production infrastructure, such as EGI, where scalability issues can be studied, and where the greater variety in resource provider configurations, for example LRMS, accelerators and network interconnects, would have been available.

vGPGPU Performance Evaluations

One notable bottleneck that emerged when studying the vGPGPU-enabled grid infrastructure was the testbed 1Gbps Cat5e Ethernet network, whose bandwidth and latency had a performance impact on applications that involved CPU-to-GPGPU data transfers. However, despite the limitations of a relatively slow network, the throughput of the mixed workloads of single vGPGPU and dual vGPGPU applications executed on the vGPGPU-enabled cluster/Compute Element was similar to their physical GPGPU equivalents.

There is some potential in pursuing further vGPGPU performance studies. For example, there may be ways to achieve greater job throughput, or there may be ways to develop vGPGPU allocation strategies that avoid bad performance. The following experiments are proposed as the basis of further research:

(i) A study of job throughput when using lower latency network interconnects such as Infiniband to examine if job throughput rates for one and two vGPGPUs exceed similar physical GPGPU job throughput rates;

(ii) There is some scope to release vGPGPUs back into the resource pool using the TORQUE/MAUI LRMS at a much earlier stage than currently done. Releasing resources earlier allows waiting vGPGPU jobs to scheduled quicker, and therefore improves vGPGPU job throughput. The current TORQUE/MAUI implementation uses a looping *Monitor* process that provides a vGPGPU *garbage collection* service. During each cycle of the Monitor process all the vGPGPU resources of completed jobs are released back into the resource pool. Once this is done, a subsequent sub-process will attempt to assign vGPGPUs to waiting jobs. This process is lock free, and avoids jobs requesting and releasing vGPGPUs at the same time. An architecture that allows jobs to asynchronously release vGPGPU resources back into the pool would require greater care to avoid any resource locking problems due to asynchronous allocation requests from the Monitor.

(iii) A similar problem to the previous resource release problem occurs when the LRMS uses a semaphore to manage floating generic consumable resources (as exemplified in the SLURM LRMS case-study). As consumable resource semaphores are not updated until the job has fully completed, no new vGPGPU will run until the semaphore has a positive value. There may be some potential job throughput improvements if the use of the LRMS semaphore can be avoided, and the resource management is handled asynchronously through the Registry.

(iv) An examination of the performance and throughput of jobs when there is greater variety in the workload mix is required. This variety includes mixing in non-GPGPU jobs with either GPGPU or vGPGPU jobs, executing multi-core/multi-GPGPU eScience applications, as well as executing a greater range of applications with different CPU-to-GPGPU communication patterns. This examination is expected to determine whether, under normal workload conditions, non-GPGPU jobs inadvertently block the execution of GPGPU jobs because they compete for the same set of CPU resources on worker nodes. Efficiency gains should be expected with vGPGPU workloads as the vGPGPU jobs can execute on any worker node. This hypothesis would need to be tested.

It should be noted that, while it would have been preferable to pursue these proposed experiments within the thesis, the time available to perform these studies was limited, and the scope and depth of the work is very substantial (and potentially the subject of several new publications).

5.4 Enhanced Functionality

Ranking of Resources Providers

The two-phase method developed in Chapter 3 does not apply any ranking (priority) to the matched Compute Elements. The net effect is that all Compute Elements are treated equally. However, there are scenarios where ranking the Compute Elements according to some need would be desirable. For example, if the vGPGPU resource integration case is considered, then a simple measure for establishing a Compute Element preference could be based on the ratio of unallocated vGPGPUs to the total number installed, and this is expressed by the equation:

$$GPGPURank = \frac{FreeVGPGPU_{VirtualGPGPUMethod}}{TotalVGPGPU_{VirtualGPGPUMethod}};\ where\ GPGPU\_Rank \in [0,1]$$

The closer this expression evaluates to 1, the more likelihood that the job will not have to wait for vGPGPU resources before being scheduled to run. The problem with the equation is that the GPGPURank could potentially evaluate to the same value regardless of the size of the resource pool at each grid resource provider. For example, if we assume three Compute Elements *A, B and C* have 1,10 and 50 free virtual-GPGPUs from respective pools of size 2, 20, 100, and these each have the highest GPGPU rank of 0.5 out of all of the Compute Elements, then a second expression that also considers the size of each Compute Element's vGPGPU resource pool could also be used to further rank these three sites in order. This scenario could be expressed by the supplemental equation:

$$GPGPURankMaxFree = GPGPURank \times \frac{FreeVGPGPU_{VirtualGPGPUMethod}}{Max(TotalVGPGPU_{VirtualGPGPUMethod})}$$

However, the consequence for the two-phase submission process is that this expression requires first determining the largest resource vGPGPU pool available for the user's job.

Extending the Registry for Other Resources

The Registry is currently designed to handle a single type of resource – the virtual-GPGPU. In order to extend the Registry so that other types of floating resources (e.g.

FPGAs over RPC) may be used would require some changes to the Registry service.

At present the vGPGPU Registry service is managed by a single database table and protocol. It is likely that each new resource type will be best served by its own database schema and table, rather than a single homogeneous table and access protocol. If the multiple table approach is taken, then the protocol for handling resource querying, registering and allocation would need to be adapted. The question of whether that adaptation should take the generic form

$$Query(ResourceType_a, \ldots)$$

or, whether it should be specific to the resource

$$QueryResourceType_a(\ldots)$$

would be one of the technical matters that needs careful consideration.

Per Virtual Organisation Views

Fair-shares are LRMS policies that help manage the allocation of CPU resources on a per user or groups of users basis. The grid middleware and GLUE Schema support the use of fair-shares through "VO views". The aim is to allow users and workload management systems decide on the placement of jobs based on the how many CPUs (or, for storage, disk space) are available to the VO when any resource sharing polices of a Compute Element (or indeed an LRMS) are defined. It helps avoid, for example, the placement of jobs at resource centres where the VO has exhausted their share of resources.

At present, there is no support for implementing CDER or virtual-CDER fair-shares on an LRMS or, by extension, on a Grid. Solving this problem is non-trivial. Firstly, fair-share policies are applied to collections of worker nodes (e.g. Queues) over a finite period of time. They do not apply to the individual nodes themselves, and certainly not to individual CPUs or CDERs. Secondly, there is very little support for monitoring the usage of individual CDERs. For example, some high-end NVIDIA GPGPU models have support for measuring usage, but this does not apply to all NVIDIA GPGPUs, and certainly not for all CDERs.

Handling Offline Resources

None of the prototypes handle cases when a specific worker node or nodes providing vGPGPU CDERs (the physical node or the container) need to be marked offline for maintenance reasons. The non-availability of a resource effects the quality and veracity of published information, which can influence where a grid job executes. In the case of CDERs (e.g GPGPUs) the grid information provider needs to examine if a worker node providing the CDER is offline and make adjustments to calculation of the free unallocated CDERs. In the case of virtual-CDERs, the Registry must be able to handle marking each of its containers offline individually or marking the container's VM host offline. Resources should only be allocated from the online pool of CDERs. Provisions for such cases need to be available for production systems.

5.5 Potential Research and Development Directions

Recommender Systems for Resource Brokering

The HTCondor ClassAd has been central to resource brokering on Grids, matching grid jobs to suitable resource providers and their resources. The broker works by converting a snapshot of the information from the grid information system (e.g. a Top-level BDII) into ClassAd Machine Resource Offers. These offers are then compared to the requirements in the grid job description.

Each job submitted through a workload management system will evaluate the suitability of potential resources. This evaluation is based on the latest available copy of the grid information. At no stage is information about the historical performance of the resource providers or individual resources recorded and maintained, so there is a risk that jobs could be targeted at resources that intermittently fail.

Recommender Systems (also known as "Recommendation Systems") are services that analyse user requirements and preferences to determine potential resources that a user (or customer) may be interested in using in the future. The Recommender System processes information about each resource and ranks them in order according to the user's specific requirements. For example, the classic Recommender System is the Google web-search service. The service processes a user specified search term, and returns a list of matching results. The ordering of the matched items will often

be determined by a number of external factors, such as the popularity of particular resources (e.g have previous users selected this item before?). Similarly, the video streaming content provider Netflix will make recommendations based on the customer's previous viewing history, the historical choices of other customers with similar interests, items that are of global interest, and other customers ratings.

The addition of a Recommender System to existing Grids could be beneficial for the following reasons:

(i) The user can use the Recommender System as a pre-filter to discovery resources using simple, fuzzy search terms (e.g. +GPGPU +CUDA).

(ii) The data returned by this pre-filter could be ranked based on historical rating of the resource.

The use of customer ratings as a measure of product or service satisfaction has the potential to be re-applied as a measure of the quality of service delivered by each resource. Resources that consistently fail (using resource monitoring tools) could be assigned a low rank, making them less likely for selection (or potentially blacklisted), while resources that do not fail and exhibit good performance can improve their rating. As it currently stands, Grid resource brokers do not use any quality of service metric to determine the fitness of a resource centre to execute a job.

5.6 Conclusions

This thesis investigated whether a conceptual approach to the integration of accelerated computing resources into large multi-disciplinary computational grid infrastructures could be used to address the problem of integrating a wide range of new resources. Such Grids are constrained in how they can adapt to changes, as they require operational and (user) environmental stability. The need to integrate new resources versus the need to ensure operational and environmental stability was referred to as the Resource Integration Problem.

An approach was proposed that set out a basic set of requirements that the integration of any resource into the grid ought to follow. By following these requirements, it was shown how to develop concrete solutions that integrated both GPGPU and virtual-GPGPUs – resource types that are managed and handled very differently. Furthermore, the manner in which the new resources were integrated required only a few

unobtrusive minor changes to the grid middleware, and required no changes to the grid standards. This was made possible by the development of new tools and services that hooked into existing grid services or interacted with them.

It is the author's belief that the proposed Conceptual Approach is both successful and an appropriate methodology for tackling the Resource Integration Problem.

Appendices

Appendix A

Resource Deployment Data

To extract state data from the BDII and determine information about the number of resource providers, ComputeElements and ComputerElement queues, the following ldapsearch commands were used:

Listing A.1: ldapsearch commands of a BDII to determine key resource information

```
# Calculate the number of resource providers
ldapsearch -x -LLL -h  lcg-bdii.cern.ch -o ldif-wrap=no -p 2170 -b "o=
   ↪ grid"  GlueCEUniqueID GlueSiteName | grep GlueSiteName: | sort |
   ↪  uniq | wc -l

# Calculate the number of ComputeElements
ldapsearch -x -LLL -h  lcg-bdii.cern.ch -o ldif-wrap=no -p 2170 -b "o=
   ↪ grid"  GlueClusterUniqueID=* GlueSiteUniqueID | grep dn: | sort
   ↪ | uniq | wc -l


# Calculate the number of Queues
ldapsearch -x -LLL -h  lcg-bdii.cern.ch -o ldif-wrap=no -p 2170 -b "o=
   ↪ grid"  '(GlueCEUniqueID=*)' GlueCEUniqueID | grep GlueCEUniqueID
   ↪  | sort | uniq | wc -l
```

Appendix B

Accounting for Resource Usage

Accounting information for individual hardware devices is not always available, and therefore it may not be possible to report usage information back to the LRMS. However, some Operating Systems do provide tools that allow the system to audit individual operations on any file (including hardware devices), such as operations to open and close the device. An example of auditing policy that records any file open system call on an NVIDIA GPGPU /dev/nvidia0 under Linux using *auditd* is listed below.

Listing B.1: Recording an file open operation on an NVIDIA GPGPU

```
/etc/init.d./auditd start
auditctl -w   /dev/nvidia0  -S open -k nvidia0

/usr/local/cuda-7.5/samples/1_Utilities/deviceQuery

ausearch -k nvidia0
----
time->Wed Nov  4 16:37:55 2015
type=CONFIG_CHANGE msg=audit(1446655075.226:5): auid=0 ses=470 subj=
   ↪ unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 op="add
   ↪ rule" key="nvidia0" list=4 res=1
----
time->Wed Nov  4 16:38:03 2015
type=PATH msg=audit(1446655083.854:6): item=0 name="/dev/nvidia0"
   ↪ inode=553014 dev=00:05 mode=020666 ouid=0 ogid=0 rdev=c3:00 obj=
   ↪ unconfined_u:object_r:device_t:s0 nametype=NORMAL
type=CWD msg=audit(1446655083.854:6):  cwd="/usr/local/cuda-7.5/
   ↪ samples/1_Utilities/deviceQuery"
type=SYSCALL msg=audit(1446655083.854:6): arch=c000003e syscall=2
   ↪ success=yes exit=4 a0=7ffdb3fec0c0 a1=2 a2=7ffdb3fec0cc a3=0
   ↪ items=1 ppid=9914 pid=10326 auid=0 uid=0 gid=0 euid=0 suid=0
   ↪ fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts1 ses=472 comm="deviceQuery
   ↪ " exe="/usr/local/cuda-7.5/samples/1_Utilities/deviceQuery/
   ↪ deviceQuery" subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:
   ↪ c0.c1023 key="nvidia0"
```

Accounting for resource usage in this manner will require the development of au-

diting policies, as well as additional tools to publish and collate each of the events into a central repository. After publishing, new tools would be required to analyse those events and establish correlations between those events and jobs in the LRMS.

Appendix C

LRMS GPGPU Utilisation Function

The utilisation is measured as the ratio of the total time GPGPUs are allocated to jobs during the complete processing of an entire workload mix and the total number of seconds that the workload mix executed on all of the GPGPUs.

An example of the calculation is as follows: during the execution of "Mix 25 75" on a pGPGPU deployment 25 jobs each used 1 GPGPU and executed for an average of 47.18 second, while 75 jobs used 2 GPGPUs and executed for 48.58 seconds. The total time that the jobs were allocated GPGPUs for is:

$$25 \times 1 \times 47.18 + 75 \times 2 \times 48.58 = 8466.5\ GPGPU seconds$$

However, the workload mix used 10 GPGPUs[1] for 1216.8 seconds, i.e. it consumed a total of

$$10 \times 1216.8 = 12168\ GPGPU seconds$$

The utilisation is calulated as

$$Utilisation_{Mix\ 25\ 75} = \frac{8466.5}{12168} = 0.6958004602$$

The generalised formula is defined as:

$$U_{Mix\_X\_Y} = \frac{X \times AvgTime_X + 2 \times Y \times AvgTime_Y}{TotalTime_{Mix\_X\_Y} \times GPGPUPoolSize}$$

[1] the number of GPGPUs in the testbed

Bibliography

[1] Ian Foster and Carl Kesselman, eds. *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. ISBN: 1-55860-475-8 (cit. on pp. 1, 43).

[2] Alison Wright. "Nobel Prize 2013: Englert and Higgs". In: *Nature Physics* 9.11 (Nov. 2013). Research Highlights, pp. 692–692. ISSN: 1745-2473. URL: http://dx.doi.org/10.1038/nphys2800 (cit. on pp. 1, 52).

[3] Ian Bird. "Computing for the Large Hadron Collider". In: *Annual Review of Nuclear and Particle Science* 61 (2014), pp. 99–118. DOI: 10.1146/annurev-nucl-102010-130059. URL: http://www.annualreviews.org/doi/abs/10.1146/annurev-nucl-102010-130059 (cit. on pp. 1, 17, 52).

[4] S.H. Fuller and L.I. Millet, eds. *The Future of Computing Performance: Game Over or Next Level?* National Research Council of the National Academies, 2012 (cit. on pp. 1, 32).

[5] *Top 500 Supercomputers Highlights – June 2015*. URL: http://www.top500.org/lists/2015/06/highlights/ (visited on 08/15/2015) (cit. on pp. 1, 32).

[6] R. Chandra et al. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2001 (cit. on p. 1).

[7] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface,* 2nd edition. Cambridge, MA: MIT Press, 1999 (cit. on p. 1).

[8] Enol Fernandez del Castillo, John Walsh, and Alvaro Simon. "Parallel Computing Workshop". In: *Proceedings of the EGI Community Forum 2012/EMI Second Technical Conference*. http://pos.sissa.it/archive/conferences/162/057/EGICF12-EMITC2_057.pdf. Munich, Germany: Proceedings of Science, Mar. 2012 (cit. on pp. 2, 81).

[9] John Walsh et al. “Results from the EGI GPGPU Virtual Team’s User and Resource Centre Administrators Surveys”. In: *Crakow Grid Workshop 2012*. Crakow, Poland, Oct. 2012 (cit. on p. 2).

[10] EGI-Engage Project. *EGI-Engage Accelerated Computing Task*. May 2015. URL: https://wiki.egi.eu/wiki/EGI-Engage:TASK_JRA2.4_Accelerated_Computing (visited on 10/24/2015) (cit. on p. 2).

[11] Enol Fernandez-del-Castillo, Diego Scardaci, and Alvaro Lopez Garcia. “The EGI Federated Cloud e-Infrastructure”. In: *Procedia Computer Science* 68 (2015). 1st International Conference on Cloud Forward: From Distributed to Complete Computing, pp. 196–205. ISSN: 1877-0509. DOI: http://dx.doi.org/10.1016/j.procs.2015.09.235. URL: http://www.sciencedirect.com/science/article/pii/S187705091503080X (cit. on p. 3).

[12] John Walsh, Brian Coghlan, and David O’Callaghan. “Supporting grid-enabled GPU workloads using rCUDA and StratusLab”. In: *Proceedings of Science, EGI Community Forum 2012 / EMI Second Technical Conference*. Munich, Germany: http://pos.sissa.it/archive/conferences/162/010/EGICF12-EMITC2_010.pdf, 2012 (cit. on p. 6).

[13] John Walsh et al. “Overview and Evaluation of Conceptual Strategies for Accessing CPU-Dependent Execution Resources in Grid Infrastructures”. In: *Computer Science Journal*. Ed. by Jacek Kitowski. Vol. 16. 4. Accepted for Publication. Krakow, Poland: AGH University of Science and Technology Press, 2015 (cit. on pp. 6, 7, 9).

[14] John Walsh and Jonathan Dukes. “Supporting job-level secure access to GPGPU resources on existing grid infrastructures”. In: *Proc. of the Federated Conference on Computer Science and Information Systems*. Ed. by Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki. Warsaw, Poland, Sept. 2014, pp. 781–790. ISBN: 978-83-60810-58-3. DOI: 10.15439/2014F337. URL: http://dx.doi.org/10.15439/2014F337 (cit. on pp. 7, 9, 28, 82, 96).

[15] S. Burke et al. *The gLite 3.2 User Guide*. https://edms.cern.ch/file/722398/1.4/gLite-3-UserGuide.pdf. July 2012 (cit. on pp. 7, 45, 54).

[16] John Walsh and Jonathan Dukes. “GPGPU Virtualisation with Multi-API Support using Containers”. English. In: *Euro-Par 2015: Parallel Processing Work-*

shops. Ed. by Sascha Hunold et al. Vol. 9523. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 802–812. ISBN: 978-3-319-27307-5. DOI: 10.1007/978-3-319-27308-2_64. URL: http://dx.doi.org/10.1007/978-3-319-27308-2_64 (cit. on pp. 7, 10).

[17] John Walsh and Jonathan Dukes. "Application Support for Virtual GPGPUs in Grid Infrastructures". In: *e-Science (e-Science), 2015 IEEE 11th International Conference on*. Aug. 2015, pp. 67–77. DOI: 10.1109/eScience.2015.45 (cit. on pp. 7, 10).

[18] T. Doherty et al. "Many-core on the Grid: From exploration to production". In: *Journal of Physics: Conference Series* 513.5 (2014), p. 052037. URL: http://stacks.iop.org/1742-6596/513/i=5/a=052037 (cit. on p. 7).

[19] *Torque Resource Manager*. http://www.adaptivecomputing.com/products/open-source/torque/ (cit. on pp. 15, 132).

[20] *SLURM Batch Scheduler Documentation*. http://www.schedmd.com/slurmdocs/ (cit. on p. 15).

[21] *IBM Platform Computing LSF Webpage*. http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/index.html (cit. on p. 15).

[22] *Univa Grid Engine Version 8*. http://www.univa.com/products/grid-engine/whats-new (cit. on p. 15).

[23] Douglas Thain, Todd Tannenbaum, and Miron Livny. "Distributed Computing in Practice: The Condor Experience: Research Articles". In: *Concurr. Comput. : Pract. Exper.* 17.2-4 (Feb. 2005), pp. 323–356. ISSN: 1532-0626. DOI: 10.1002/cpe.v17:2/4. URL: http://dx.doi.org/10.1002/cpe.v17:2/4 (cit. on pp. 15, 67).

[24] N. Capit et al. "A batch scheduler with high level components". In: *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*. Vol. 2. May 2005, pp. 776–783. DOI: 10.1109/CCGRID.2005.1558641 (cit. on p. 15).

[25] *Maui Cluster Scheduler Webpage*. http://www.adaptivecomputing.com/resources/docs/maui/index.php (cit. on p. 16).

[26] *Moab Documentation Website* (cit. on pp. 16, 95).

[27] Volodymy Kindratenko. *Computational Accelerator Term Revisted*. https://www.ieeetcsc.org/activities/blog/Accelerated_Computing_computational_Accelerator_Term_Revisited. Feb. 2013 (cit. on p. 18).

[28] Brendan Gregg. *Systems Performance: Enterprise and the Cloud*. Prentice Hall. Morgan Kaufmann Publications Inc., 2013 (cit. on p. 19).

[29] Michael Flynn. "Some Computer Organizations and Their Effectiveness". In: *Computers, IEEE Transactions on* C-21.9 (Sept. 1972), pp. 948–960. ISSN: 0018-9340. DOI: 10.1109/TC.1972.5009071 (cit. on p. 19).

[30] Dezso Sima, Terence Fountain, and Peter Kacsuk. *Advanced Computer Architectures: A Design Space Approach*. Pearson Education, 1997. ISBN: 9788131702086 (cit. on p. 19).

[31] Gerassimos Barlas. *Multicore and {GPU} Programming: An Integrated Approach*. Boston: Morgan Kaufmann, 2015. ISBN: 978-0-12-417137-4. DOI: http://dx.doi.org/10.1016/B978-0-12-417137-4.09994-9 (cit. on pp. 19, 20).

[32] J. Jeffers and J. Reinders. *Xeon Phi Coprocessor High Performance Programming*. Morgan Kaufmann Publishers, 2013 (cit. on p. 22).

[33] D. Kirk and W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers, 2010 (cit. on p. 25).

[34] M. Scarpino. *OpenCL in Action: How to accelerate graphics and computation*. Manning Publications Co., 2011 (cit. on p. 25).

[35] Andrew Kerr, Gregory Diamos, and Sudhakar Yalamanchili. "GPU Application Development, Debugging, and Performance Tuning with GPU Ocelot". In: *GPU Computing GEMS Jade Edition, 1st Edition*. Ed. by Wen-mei Hwu et al. Applications of GPU Computing Series. Morgan Kaufmann, 2012. Chap. 30. ISBN: 978-0-12-385963-1. DOI: http://dx.doi.org/10.1016/B978-0-12-385963-1.00030-7. URL: http://www.sciencedirect.com/science/article/pii/B9780123859631000307 (cit. on p. 25).

[36] *The Altera SDK for OpenCL*. URL: https://www.altera.com/en_US/pdfs/literature/hb/opencl-sdk/aocl_programming_guide.pdf (visited on 09/10/2015) (cit. on p. 26).

[37] *Rosetta Stone of Workload Managers*. URL: http://slurm.schedmd.com/rosetta.html (visited on 09/13/2015) (cit. on p. 28).

[38] *Comparison of cluster software*. URL: https://en.wikipedia.org/wiki/Comparison_of_cluster_software (visited on 09/13/2015) (cit. on p. 28).

[39] *Batch System Comparison*. URL: https://twiki.cern.ch/twiki/bin/view/LCG/BatchSystemComparison (visited on 09/13/2015) (cit. on p. 28).

[40] C. Cavazzonia et al. "Resource Scheduling Best Practice in Hybrid Clusters". In: (2013). URL: http://www.prace-ri.eu/IMG/pdf/WP148.pdf (visited on 09/15/2015) (cit. on p. 28).

[41] Jerry Eriksson et al. "PRACE Implementation Phase 2 Deliverable D11.2: System Software and Application Development Environments". In: (2014). URL: http://www.prace-ri.eu/IMG/pdf/D11.2_2ip.pdf (visited on 09/16/2015) (cit. on p. 28).

[42] *Intel Xeon Phi: Designating Available Coprocessors*. URL: https://software.intel.com/en-us/node/540596 (visited on 09/13/2015) (cit. on p. 29).

[43] Adam DeConinck. *Tools and Tips for Managing a GPU Cluster*. URL: http://on-demand.gputechconf.com/gtc/2014/presentations/S4253-tools-tips-for-managing-a-gpu-cluster.pdf (visited on 12/28/2015) (cit. on p. 29).

[44] *AMD APP OpenCL Programming Guide*. http://developer.amd.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide-rev-2.7.pdf. 2014 (cit. on p. 29).

[45] *TORQUE NVIDIA GPGPUs*. URL: http://docs.adaptivecomputing.com/torque/4-0-2/Content/topics/3-nodes/NVIDIAGPGPUs.htm (visited on 12/28/2015) (cit. on p. 29).

[46] *Scheduling GPUs*. URL: http://docs.adaptivecomputing.com/torque/4-0-2/Content/topics/3-nodes/schedulingGPUs.htm (visited on 12/28/2015) (cit. on p. 29).

[47] *Top 500 Supercomputers*. URL: http://www.top500.org/ (visited on 08/15/2015) (cit. on p. 30).

[48] Jack Dongarra. "The LINPACK Benchmark: An Explanation". In: *Proceedings of the 1st International Conference on Supercomputing*. London, UK, UK: Springer-Verlag, 1988, pp. 456–474. ISBN: 3-540-18991-2. URL: http://dl.acm.org/citation.cfm?id=647970.742568 (cit. on p. 30).

[49] M Christon et al. “ASCI RED-experiences and lessons learned with a massively parallel teraflop supercomputer”. In: *Proceedings of the Supercomputer 1997* (1997) (cit. on p. 30).

[50] Danielle Venton. *Playstation goes from games to grid*. http://www.isgtw.org/feature/feature-playstation-goes-games-grid. Feb. 2010 (cit. on pp. 30, 55).

[51] Yuri Nishikawa et al. “Performance Analysis of ClearSpeed’s CSX600 Interconnects”. In: *Parallel and Distributed Processing with Applications, 2009 IEEE International Symposium on*. Aug. 2009, pp. 203–210. DOI: 10.1109/ISPA.2009.102 (cit. on p. 30).

[52] Guido Juckeland et al. “SPEC ACCEL: A Standard Application Suite for Measuring Hardware Accelerator Performance”. English. In: *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*. Ed. by Stephen A. Jarvis, Steven A. Wright, and Simon D. Hammond. Vol. 8966. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 46–67. ISBN: 978-3-319-17247-7. DOI: 10.1007/978-3-319-17248-4_3. URL: http://dx.doi.org/10.1007/978-3-319-17248-4_3 (cit. on pp. 35, 135).

[53] John A. Stratton et al. *Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing*. Tech. rep. IMPACT-12-01. University of Illinois at Urbana-Champaign, Mar. 2012. URL: http://impact.crhc.illinois.edu/shared/docs/impact-12-01.parboil.pdf (visited on 08/15/2015) (cit. on p. 35).

[54] Shuai Che et al. “Rodinia: A benchmark suite for heterogeneous computing”. In: *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Oct. 2009, pp. 44–54. DOI: 10.1109/IISWC.2009.5306797 (cit. on p. 35).

[55] *IEEE Xplore Digital Library*. URL: http://ieeexplore.ieee.org/Xplore/home.jsp (visited on 12/27/2015) (cit. on p. 35).

[56] *Google Scholar*. URL: https://scholar.google.com/ (visited on 12/27/2015) (cit. on p. 35).

[57] Anthony Danalis et al. “The Scalable Heterogeneous Computing (SHOC) benchmark suite”. In: *Proceedings of 3rd Workshop on General Purpose Processing on Graphics Processing Units*. Ed. by David R. Kaeli and Miriam Leeser. Vol. 425. ACM International Conference Proceeding Series. Pittsburgh, PA, USA: ACM, Mar. 2010, pp. 63–74. ISBN: 978-1-60558-935-0. DOI: 10.1145/1735688.1735702. URL: http://doi.acm.org/10.1145/1735688.1735702 (cit. on pp. 35, 40, 135).

[58] *Current SPEC Benchmark Pricing*. URL: https://www.spec.org/order.html (visited on 08/20/2015) (cit. on p. 36).

[59] P. Mell and T. Grance. *The NIST Definition of Cloud Computing, NIST Special Publication 800-145*. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. 2011 (cit. on p. 36).

[60] *Meridian Technology Centre: Nvidia VDI with Nvidia GRID case-study*. http://international.download.nvidia.com/pdf/grid/resources/nvidia-grid-case-study-meridian.pdf (cit. on p. 39).

[61] *GPUBox Administration Guide and Reference*. Mar. 2015. URL: http://www.renegatt.com/download.php?GPUBox_Admin_Guide_and_Reference.pdf (visited on 10/23/2015) (cit. on p. 39).

[62] Antonio J. Peña et al. “A complete and efficient CUDA-sharing solution for HPC clusters”. In: *Parallel Computing* 40.10 (2014), pp. 574–588. DOI: 10.1016/j.parco.2014.09.011. URL: http://dx.doi.org/10.1016/j.parco.2014.09.011 (cit. on pp. 39, 40, 135).

[63] Tyng-Yeu Liang and Yu-Wei Chang. “GridCuda: A Grid-Enabled CUDA Programming Toolkit”. In: *25th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. Biopolis, Singapore, Mar. 2011, pp. 141–146. DOI: 10.1109/WAINA.2011.82. URL: http://dx.doi.org/10.1109/WAINA.2011.82 (cit. on pp. 39, 142).

[64] Lin Shi, Hao Chen, and Jianhua Sun. “vCUDA: GPU accelerated high performance computing in virtual machines”. In: *23rd IEEE International Symposium on Parallel and Distributed Processing, (IPDPS)*. Rome, Italy, May 2009, pp. 1–11. DOI: 10.1109/IPDPS.2009.5161020. URL: http://dx.doi.org/10.1109/IPDPS.2009.5161020 (cit. on p. 39).

[65] Giulio Giunta et al. “A GPGPU Transparent Virtualization Component for High Performance Computing Clouds”. English. In: *Proceedings of the 16th International Euro-Par Conference (Euro-Par'10)*. Vol. 6271. LNCS. Sept. 2010, pp. 379–391. ISBN: 978-3-642-15276-4. DOI: 10.1007/978-3-642-15277-1_37. URL: http://dx.doi.org/10.1007/978-3-642-15277-1_37 (cit. on p. 39).

[66] Amnon Barak and Amnon Shiloh. “The Virtual OpenCL (VCL) Cluster Platform”. In: *Proc. Intel European Research & Innovation Conf.* 2011, p. 196 (cit. on pp. 39, 40).

[67] Philipp Kegel, Michel Steuwer, and Sergei Gorlatch. “dOpenCL: Towards uniform programming of distributed heterogeneous multi-/many-core systems.” In: *J. Parallel Distrib. Comput.* 73.12 (2013), pp. 1639–1648. DOI: http://dx.doi.org/10.1016/j.jpdc.2013.07.021. URL: http://dblp.uni-trier.de/db/journals/jpdc/jpdc73.html#KegelSG13 (cit. on p. 40).

[68] Jungwon Kim et al. “SnuCL: an OpenCL framework for heterogeneous CPU/GPU clusters”. In: *International Conference on Supercomputing (ICS'12)*. Venice, Italy, June 2012, pp. 341–352. DOI: 10.1145/2304576.2304623. URL: http://doi.acm.org/10.1145/2304576.2304623 (cit. on p. 40).

[69] C.J. Reynolds, Z. Lichtenberger, and S. Winter. “Provisioning OpenCL Capable Infrastructure with Infiniband Verbs”. In: *10th International Symposium on Parallel and Distributed Computing (ISPDC)*. July 2011, pp. 110–116. DOI: 10.1109/ISPDC.2011.25 (cit. on p. 40).

[70] Steve Plimpton. “Fast Parallel Algorithms for Short-Range Molecular Dynamics”. In: *Journal of Computational Physics* 117.1 (1995), pp. 1–19. ISSN: 0021-9991. DOI: DOI:10.1006/jcph.1995.1039 (cit. on p. 40).

[71] John Paul Walters et al. “GPU Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications”. In: *IEEE 7th International Conference on Cloud Computing*. Anchorage, AK, USA: IEEE, June 2014, pp. 636–643. ISBN: 978-1-4799-5063-8. DOI: 10.1109/CLOUD.2014.90. URL: http://dx.doi.org/10.1109/CLOUD.2014.90 (cit. on pp. 40, 135).

[72] Flavio Vella et al. “GPU Computing in EGI Environment Using a Cloud Approach”. In: *International Conference on Computational Science and Its Appli-*

cations (ICCSA). Santander, Spain, June 2011, pp. 150–155. DOI: 10.1109/ICCSA.2011.61. URL: http://doi.ieeecomputersociety.org/10.1109/ICCSA.2011.61 (cit. on pp. 40, 67, 71, 80, 142).

[73] Peter Mell and Timothy Grance. *The NIST Definition of Cloud Computing*. Apr. 2012. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf (visited on 11/12/2015) (cit. on p. 41).

[74] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. OSDI'04. San Francisco, CA: USENIX Association, 2004, pp. 10–10. URL: http://dl.acm.org/citation.cfm?id=1251254.1251264 (cit. on p. 41).

[75] *Amazon Elastic Compute Cloud User Guide for Linux Instances: Linux GPU Instances*. URL: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using_cluster_computing.html (visited on 12/29/2015) (cit. on p. 42).

[76] Abhishek Gupta et al. "The Who, What, Why, and How of High Performance Computing in the Cloud". In: *IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom 2013*. 2013, pp. 306–314. DOI: 10.1109/CloudCom.2013.47. URL: http://dx.doi.org/10.1109/CloudCom.2013.47 (cit. on p. 42).

[77] *The UberCloud Experiment*. 2015. URL: https://www.theubercloud.com/hpc-experiment/ (visited on 11/12/2015) (cit. on p. 42).

[78] *The UberCloud Teams (application case-studies)*. 2015. URL: https://www.theubercloud.com/teams/ (visited on 11/12/2015) (cit. on p. 42).

[79] Nathan Regola and Jean-Christophe Ducom. "Recommendations for Virtualization Technologies in High Performance Computing". In: *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*. CLOUDCOM '10. 2010, pp. 409–416. ISBN: 978-0-7695-4302-4. DOI: 10.1109/CloudCom.2010.71. URL: http://dx.doi.org/10.1109/CloudCom.2010.71 (cit. on p. 42).

[80] *The OpenStack Project homepage*. http://www.openstack.org (cit. on p. 42).

[81] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. "IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures". In: *IEEE Computer*. Vol. 45. Dec. 2012, pp. 65–72 (cit. on p. 42).

[82] Stephen Myers. "The Large Hadron Collider 2008–2013". In: *International Journal of Modern Physics A* 28.25 (2013). DOI: 10.1142/S0217751X13300354. URL: http://www.worldscientific.com/doi/abs/10.1142/S0217751X13300354 (cit. on p. 43).

[83] P.J. Quinn et al. "Delivering SKA Science". In: *Proceedings of Advancing Astrophysics with the Square Kilometre Array (AASKA14)*. Giardiana Naxos, Italy, June 2014. URL: http://pos.sissa.it/archive/conferences/215/147/AASKA14_147.pdf (cit. on p. 43).

[84] Simson Garfinkel and Harold Abelson. *Architects of the Information Society: 35 Years of the Laboratory for Computer Science at MIT*. Cambridge, MA, USA: MIT Press, 1999. ISBN: 0262571315 (cit. on p. 44).

[85] I. Foster et al. "A security architecture for computational grids". In: *Proceedings of the 5th ACM conference on Computer and communications security*. 1998, pp. 83–92 (cit. on p. 44).

[86] Ian T. Foster, Carl Kesselman, and Steven Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". In: *IJHPCA* 15.3 (2001), pp. 200–222. DOI: 10.1177/109434200101500302. URL: http://dx.doi.org/10.1177/109434200101500302 (cit. on p. 44).

[87] Heinz Stockinger. "Defining the grid: a snapshot on the current view". In: *The Journal of Supercomputing* 42.1 (2007), pp. 3–17. DOI: 10.1007/s11227-006-0037-9. URL: http://dx.doi.org/10.1007/s11227-006-0037-9 (cit. on p. 44).

[88] Ian Foster et al. "Cloud Computing and Grid Computing 360-Degree Compared". In: *Grid Computing Environments Workshop, GCE '08*. Nov. 2008, pp. 1–10. DOI: 10.1109/GCE.2008.4738445 (cit. on p. 44).

[89] David P. Anderson. "BOINC: A System for Public-Resource Computing and Storage". In: *5th International Workshop on Grid Computing (GRID 2004)*. Ed. by Rajkumar Buyya. Pittsburgh, PA, USA: IEEE Computer Society, Nov.

2004, pp. 4–10. ISBN: 0-7695-2256-4. DOI: 10.1109/GRID.2004.14. URL: http://doi.ieeecomputersociety.org/10.1109/GRID.2004.14 (cit. on p. 44).

[90] *User Manual for ARC 11.05 (client version 1.0.0) and above*. http://www.nordugrid.org/documents/arc-ui.pdf. 2014 (cit. on pp. 45, 49, 109).

[91] Dietmar Erwin and David Snelling. "UNICORE: A Grid Computing Environment". English. In: *Euro-Par 2001 Parallel Processing*. Ed. by Rizos Sakellariou et al. Vol. 2150. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 825–834. ISBN: 978-3-540-42495-6. DOI: 10.1007/3-540-44681-8_116. URL: http://dx.doi.org/10.1007/3-540-44681-8_116 (cit. on p. 45).

[92] *The Globus Resource Specification Language RSL v1.0*. URL: http://toolkit.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html (visited on 12/29/2015) (cit. on p. 46).

[93] Fabrizio Pacini. *Job Description Language (JDL) Attributes Specification (Submission through the WMProxy Service)*. https://edms.cern.ch/document/590869/1 (cit. on p. 46).

[94] *Job Submission Description Language (JSDL) Specification, Version 1.0*. Nov. 2005 (cit. on p. 46).

[95] *IGE BesGRAM: OGSA-BES support under GRAM 5*. http://www.ige-project.eu/ige-showcases/besgram. 2013 (cit. on p. 46).

[96] *Extended Resource Specification Language: Reference Manual for ARC versions 0.8 and above*. http://www.nordugrid.org/documents/xrsl.pdf. Mar. 2015 (cit. on p. 48).

[97] A. Dorigo et al. *EMI User's Guide: WMS command line interface guide*. http://web2.infn.it/gLiteWMS/images/WMS/Docs/wmproxy-guide.pdf (cit. on p. 49).

[98] Gabriele Garzoglio et al. "ReSS: A Resource Selection Service for the Open Science Grid". English. In: *Grid Computing*. Ed. by SimonC. Lin and Eric Yen. Springer US, 2009, pp. 89–98. ISBN: 978-0-387-78416-8. DOI: 10.1007/978-0-387-78417-5_8. URL: http://dx.doi.org/10.1007/978-0-387-78417-5_8 (cit. on p. 49).

[99] Adrian Casajus et al. “DIRAC pilot framework and the DIRAC Workload Management System”. In: *Journal of Physics: Conference Series* 219.6 (2010), p. 062049. URL: http://stacks.iop.org/1742-6596/219/i=6/a=062049 (cit. on p. 49).

[100] T Maeno et al. “Overview of ATLAS PanDA Workload Management”. In: *Journal of Physics: Conference Series* 331.7 (2011), p. 072024. URL: http://stacks.iop.org/1742-6596/331/i=7/a=072024 (cit. on p. 49).

[101] T Maeno et al. “Evolution of the ATLAS PanDA workload management system for exascale computational science”. In: *Journal of Physics: Conference Series*. Vol. 513. 3. IOP Publishing. 2014, p. 032062. DOI: doi:10.1088/1742-6596/513/3/032062 (cit. on p. 49).

[102] I. Sfiligoi et al. “The Pilot Way to Grid Resources Using glideinWMS”. In: *Computer Science and Information Engineering, 2009 WRI World Congress on*. Vol. 2. Mar. 2009, pp. 428–432. DOI: 10.1109/CSIE.2009.950 (cit. on p. 49).

[103] Sergio Andreozzi et al. *OGF GLUE 1.3 Specification*. https://redmine.ogf.org/dmsf_files/61?download=. Jan. 2007 (cit. on pp. 51, 79).

[104] *OGF GLUE 2.0 Specification*. http://redmine.ogf.org/dmsf/glue-wg?folder_id=18 (cit. on pp. 51, 96).

[105] Paul Avery and Ian Foster. *iVDGL Annual Report for 2001 and 2002*. Tech. rep. California Institute of Technology, LIGO Laboratory – MS 18-34 and Massachusetts Institute of Technology, LIGO Laboratory – MS 16NW-145, 2002. URL: https://dcc.ligo.org/public/0027/T020078/000/T020078-00.pdf (visited on 11/26/2015) (cit. on p. 51).

[106] Stephen Burke, Laurence Field, and David Horat. “Migration to the GLUE 2.0 information schema in the LCG/EGEE/EGI production Grid”. In: *Journal of Physics: Conference Series* 331.6 (2011), p. 062004. URL: http://stacks.iop.org/1742-6596/331/i=6/a=062004 (cit. on pp. 51, 52).

[107] Stephen Burke et al. “The impact and adoption of GLUE 2.0 in the LCG/EGEE production Grid”. In: *Journal of Physics: Conference Series* 219.6 (2010), p. 062005. URL: http://stacks.iop.org/1742-6596/219/i=6/a=062005 (cit. on p. 52).

[108] O. Smirnova, M. Ellert, and D. Johansson. *ARIS and EGIIS: Installation, Configuration and Usage Manual*. http://www.nordugrid.org/documents/aris-egiis.pdf (cit. on p. 54).

[109] Andrew W. Cooke et al. "R-GMA: An Information Integration System for Grid Monitoring". In: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE – OTM Confederated International Conferences, CoopIS, DOA, and ODBASE*. Ed. by Robert Meersman, Zahir Tari, and Douglas C. Schmidt. Vol. 2888. Lecture Notes in Computer Science. Catania, Sicily, Italy: Springer, Nov. 2003, pp. 462–481. ISBN: 3-540-20498-9. DOI: 10.1007/978-3-540-39964-3_29. URL: http://dx.doi.org/10.1007/978-3-540-39964-3_29 (cit. on pp. 54, 110).

[110] Darran Nathan and Ralf Clemens. "Utilizing Reconfigurable Hardware Processors via Grid Services". In: *CoRR* cs.DC/0411050 (2004). URL: http://arxiv.org/abs/cs.DC/0411050 (cit. on p. 54).

[111] Toyokazu Takagi and Tsutomu Maruyama. "Accelerating HMMER search using FPGA Grid". In: *Applications, Tools and Techniques on the Road to Exascale Computing, Proceedings of the conference ParCo 2011, 31 August - 3 September 2011, Ghent, Belgium*. Ed. by Koen De Bosschere et al. Vol. 22. Advances in Parallel Computing. IOS Press, 2011, pp. 587–594. ISBN: 978-1-61499-040-6. DOI: 10.3233/978-1-61499-041-3-587. URL: http://dx.doi.org/10.3233/978-1-61499-041-3-587 (cit. on p. 54).

[112] A. Stephen McGough and David J. Colling. "The GRIDCC project - The GRIDCC collaboration". English. In: *2006 1st International Conference on Communication Systems Software & Middleware, Vols 1 and 2*. 1st International Conference on Communication Systems Software and Middleware, New Delhi, INDIA, JAN 08-12, 2006. IEEE, 2006, 592–595. ISBN: 978-0-7803-9574-9 (cit. on pp. 54, 71, 72).

[113] The RingGrid Project Consortium. *Ringrid: White Paper on Remote Instrumentation*. URL: http://vlab.psnc.pl/ringrid/White%20Paper%20on%20Remote%20Instrumentation_ver1-1.pdf (cit. on pp. 54, 71, 72).

[114] Davide Adami et al. "The DORII project test bed: Distributed eScience applications at work". In: *5th International ICST Conference on Testbeds and*

Research Infrastructures for the Development of Networks and Communities, TRIDENTCOM 2009, Washington, DC, USA, April 6-8, 2009. Ed. by Scott F. Midkiff and Arunita Jaekel. IEEE Computer Society, 2009, pp. 1–6. ISBN: 978-1-4244-2846-5. DOI: 10.1109/TRIDENTCOM.2009.4976247. URL: http://dx.doi.org/10.1109/TRIDENTCOM.2009.4976247 (cit. on pp. 54, 71, 72).

[115] R. Pugliese et al. "Integrating Instruments in the Grid for On-line and Off-line Processing in a Synchrotron Radiation Facility". In: *Computational Methods in Science and Technology* Vol. 15, nr 1 (2009), pp. 21–30 (cit. on p. 55).

[116] G. Bolzon et al. "Preliminary deployment of Grid-assisted oceanographic applications". In: *Advances in Geosciences* 28 (2010), pp. 39–45. DOI: 10.5194/adgeo-28-39-2010. URL: http://www.adv-geosci.net/28/39/2010/ (cit. on p. 55).

[117] Matthew Scarpino. *Programming the Cell Processor: For Games, Graphics, and Computation*. 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008. ISBN: 0136008860, 9780136008866 (cit. on p. 55).

[118] Eamonn Kenny et al. *gLite Porting to the PlayStation 3 using ETICS for electronic High Throughput Screening (eHITS)*. Poster presented at the EGEE User Forum, https://www.cs.tcd.ie/eamonn.kenny/externals/ps3-poster.pdf. Uppsala, Sweden, Apr. 2010 (cit. on pp. 55, 71).

[119] Zsolt Zsoldos et al. "eHiTS: A new fast, exhaustive flexible ligand docking system". In: *Journal of Molecular Graphics and Modelling* 26.1 (2007), pp. 198–212. ISSN: 1093-3263. DOI: http://dx.doi.org/10.1016/j.jmgm.2006.06.002. URL: http://www.sciencedirect.com/science/article/pii/S1093326306000994 (cit. on p. 55).

[120] The EGI Collaboration. *The EGI in Numbers*. http://www.egi.eu/infrastructure/operations/egi_in_numbers/index.html. 2015 (cit. on p. 57).

[121] John Walsh and Brian Coghlan. "The Grid-Ireland National Grid Infrastructure." In: *IBERGRID 4th Iberian Grid Infrastructure Conference Proceedings*. Ed. by Alberto Procena et al. Netbiblio, 2010, pp. 19–23. ISBN: 978-84-9745-549-7 (cit. on p. 65).

[122] Stephen Childs et al. “A virtual testgrid or how to replicate a national grid”. In: *In Proceedings of the EXPGRID workshop on Experimental Grid*. 2006 (cit. on p. 65).

[123] Stephen Childs et al. “A single-computer grid gateway using virtual machines”. In: *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*. Vol. 1. Mar. 2005, pp. 310–315. DOI: 10.1109/AINA.2005.65 (cit. on p. 65).

[124] Brian A. Coghlan et al. “Principles of Transactional Grid Deployment.” In: *EGC*. Ed. by Peter M. A. Sloot et al. Vol. 3470. Lecture Notes in Computer Science. Springer, 2005, pp. 88–97. ISBN: 3-540-26918-5. DOI: http://dx.doi.org/10.1007/11508380_11 (cit. on p. 65).

[125] Stuart Kenny. “A Framework for Instrument Monitoring on the Grid”. PhD thesis. Trinity College Dublin, 2006 (cit. on p. 66).

[126] Stephane Dudzinski. “An extension to grid security monitoring to incorporate host-based intrusion detection”. M.Sc Thesis. Trinity College Dublin, 2008 (cit. on p. 66).

[127] Ronan Watson. “A generic framework for grid-enabled visualisation and computational steering, and its characterisation”. PhD thesis. Trinity College Dublin, 2010 (cit. on p. 66).

[128] Simon Toth and Miroslav Ruda. “Practical Experiences with Torque Meta-Scheduling in The Czech National Grid”. In: *Computer Science Journal*. Ed. by Jacek Kitowski. Vol. 13. 2. Krakow, Poland: AGH University of Science and Technology Press, 2012, pp. 33–45. DOI: 10.7494/csci.2012.13.2.33 (cit. on pp. 67, 71).

[129] Jens Breitbart and Gaurav Khanna. “An Exploration of CUDA and CBEA for Einstein@Home”. In: *Proceedings of the 8th International Conference on Parallel Processing and Applied Mathematics: Part I*. PPAM’09. Wroclaw, Poland: Springer-Verlag, 2010, pp. 486–495. ISBN: 3-642-14389-X, 978-3-642-14389-2. URL: http://dl.acm.org/citation.cfm?id=1882792.1882851 (cit. on p. 67).

[130] Ádám Visegrádi, József Kovács, and Peter Kacsuk. "Efficient Extension of gLite VOs with BOINC Based Desktop Grids". In: *Future Gener. Comput. Syst.* 32 (Mar. 2014), pp. 13–23. ISSN: 0167-739X. DOI: 10.1016/j.future.2013.10.012. URL: http://dx.doi.org/10.1016/j.future.2013.10.012 (cit. on p. 67).

[131] *How to Manage GPUs*. https://htcondor-wiki.cs.wisc.edu/index.cgi/wiki?p=HowToManageGpus. 2014 (cit. on p. 67).

[132] *EGI GPGPU Working Group HomePage*. https://wiki.egi.eu/wiki/GPGPU-WG. 2014 (cit. on p. 68).

[133] *EGI Federated Cloud Task Force Homepage*. https://wiki.egi.eu/wiki/Fedcloud-tf. 2014 (cit. on p. 68).

[134] James Murty. *Programming Amazon Web Services*. First. O'Reilly, 2008. ISBN: 9780596515812 (cit. on p. 72).

[135] ITU. *ITU-T E.800 E.800 : Definitions of terms related to quality of service*. ITU, 2008 (cit. on p. 72).

[136] Malgorzata Krakowian. *EGI Resource Centre Operational Level Agreement*. Oct. 2014. URL: https://documents.egi.eu/public/RetrieveFile?docid=31&version=16&filename=Resource%20Centre%20OLA%20v2.2%20-%20FINAL.pdf (visited on 10/24/2015) (cit. on p. 72).

[137] Ming Jiang et al. "An APEL Tool Based CPU Usage Accounting Infrastructure for Large Scale Computing Grids". English. In: *Data Driven e-Science*. Ed. by Simon C. Lin and Eric Yen. Springer New York, 2011, pp. 175–186. ISBN: 978-1-4419-8013-7. DOI: 10.1007/978-1-4419-8014-4_14. URL: http://dx.doi.org/10.1007/978-1-4419-8014-4_14 (cit. on p. 77).

[138] Philippe Canal. "Gratia: New Challenges in Grid Accounting". In: *Journal of Physics: Conference Series* 331.6 (2011), p. 062028. URL: http://stacks.iop.org/1742-6596/331/i=6/a=062028 (cit. on p. 77).

[139] Sy Holliger. *EGI Pay-for-Use Models – Evolving EGI Workshop*. Jan. 2013. URL: http://indico.egi.eu/indico/getFile.py/access?contribId=9&sessionId=3&resId=1&materialId=slides&confId=1252 (visited on 10/21/2015) (cit. on p. 77).

[140] Wolfgang Barth. *Nagios: System and Network Monitoring*. Second. No Starch Press, 2008. ISBN: 978-1593271794 (cit. on p. 78).

[141] Cécile Germain-Renaud et al. "Scheduling for Responsive Grids". In: *Journal of Grid Computing* 6.1 (2008), pp. 15–27. DOI: 10.1007/s10723-007-9086-4. URL: http://dx.doi.org/10.1007/s10723-007-9086-4 (cit. on p. 80).

[142] *The EGI MPI Users Guide*. https://wiki.egi.eu/wiki/MPI_User_Guide (cit. on p. 81).

[143] Rajesh Raman, Miron Livny, and Marvin Solomon. "Matchmaking: Distributed Resource Management for High Throughput Computing". In: *In Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*. 1998, pp. 28–31 (cit. on pp. 84, 89).

[144] Javier Cacheiro. *Analysis of Batch Systems: SGE, SLURM, and TORQUE/-Maui*. Feb. 2014. URL: https://www.cesga.es/en/biblioteca/downloadAsset/id/753 (visited on 10/20/2015) (cit. on p. 95).

[145] *Maui Batch Scheduler System GPGPU code support*. http://www.supercluster.org/pipermail/torqueusers/2012-February/014041.html (cit. on p. 95).

[146] *Forward of Requirements To The Batch System*. URL: http://grid.pd.infn.it/cream/field.php?n=Main.ForwardOfRequirementsToTheBatchSystem (visited on 10/23/2015) (cit. on p. 98).

[147] *Tornado HomePage*. http://www.tornadoweb.org/. 2014 (cit. on p. 103).

[148] *Puppet IT Automation Homepage*. http://puppetlabs.com/. 2014 (cit. on p. 106).

[149] Salman Toor et al. *Case-Study for Different Models of Resource Brokering in Grid Systems*. 2010. DOI: 10.1.1.220.4891. URL: http://www.it.uu.se/research/reports/2010-009/2010-009-nc.pdf (cit. on p. 108).

[150] *CUDA Wrapper SourceForge Homepage*. http://sourceforge.net/projects/cudawrapper/. 2014 (cit. on p. 109).

[151] Volodymyr V. Kindratenko et al. "GPU clusters for high-performance computing". In: *Proceedings of the 2009 IEEE International Conference on Cluster Computing, August 31 - September 4, 2009, New Orleans, Louisiana, USA*. IEEE Computer Society, 2009, pp. 1–8. ISBN: 978-1-4244-5012-1. DOI: 10.1109/

CLUSTR.2009.5289128. URL: http://dx.doi.org/10.1109/CLUSTR.2009.5289128 (cit. on p. 109).

[152] Enol Fernandez del Castillo. "Support to MPI Applications on the Grid." In: *Computing and Informatics* 31.1 (2012), pp. 149–160 (cit. on p. 109).

[153] Stuart Kenny and Brian A. Coghlan. "Towards a Grid-wide Intrusion Detection System". In: *Advances in Grid Computing - EGC 2005, European Grid Conference, Amsterdam, The Netherlands, February 14-16, 2005, Revised Selected Papers*. Ed. by Peter M. A. Sloot et al. Vol. 3470. Lecture Notes in Computer Science. Springer, 2005, pp. 275–284. ISBN: 3-540-26918-5. DOI: 10.1007/11508380_29. URL: http://dx.doi.org/10.1007/11508380_29 (cit. on p. 110).

[154] Oliver Oberst et al. "Dynamic Extension of a Virtualized Cluster by using Cloud Resources". In: *Journal of Physics: Conference Series* 396.3 (2012), p. 032081. URL: http://stacks.iop.org/1742-6596/396/i=3/a=032081 (cit. on p. 122).

[155] Shrikrishna Holla. *Orchestrating Docker*. Packt Publishing, 2015. ISBN: 1783984783 (cit. on p. 123).

[156] *Pipework*. https://github.com/jpetazzo/pipework (cit. on p. 123).

[157] Sander Pronk et al. "GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit". In: *Bioinformatics* 29.7 (Feb. 13, 2013), pp. 845–854. ISSN: 1460-2059. DOI: 10.1093/bioinformatics/btt055. URL: http://dx.doi.org/10.1093/bioinformatics/btt055 (cit. on p. 138).

[158] Carlos Reaño et al. "Influence of InfiniBand FDR on the performance of remote GPU virtualization". In: *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. Indianapolis, IN, USA, Sept. 2013, pp. 1–8. DOI: 10.1109/CLUSTER.2013.6702662. URL: http://dx.doi.org/10.1109/CLUSTER.2013.6702662 (cit. on p. 141).

[159] Paul Menage, Paul Jackson, and Christoph Lameter. *CGROUPS*. 2015. URL: https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt (visited on 12/05/2015) (cit. on p. 152).

[160] Rami Rosen. *Resource management: Linux kernel Namespaces and cgroup*. 2013. URL: http://media.wix.com/ugd/295986_d5059f95a78e451db5de3d54f711e45d.pdf (visited on 12/05/2015) (cit. on p. 152).

[161] Yannis Georgiou. *Resource management with Linux Control Groups in HPC Clusters*. 2012. URL: http://slurm.schedmd.com/pdfs/LCS_cgroups_BULL.pdf (visited on 12/05/2015) (cit. on p. 152).

[162] NVIDIA Coropration. *NVIDIA CUDA TOOLKIT V7.0: Release Notes for Windows, Linux, and Mac OS*. Mar. 2015. URL: http://developer.download.nvidia.com/compute/cuda/7_0/Prod/doc/CUDA_Toolkit_Release_Notes.pdf (visited on 12/05/2015) (cit. on p. 152).