Preface

For more than a decade, the development of grid computing was driven by scientific applications. The need to solve large-scale, increasingly complex problems, motivated research on grids systems. Many interesting problems have been solved with the help of grids, for instance the nug30 Quadratic Assignment Problem.

This challenging optimization problem was posed in 1968 and requires, given a set of *n* facilities, a set of *n* locations, a distance specified for each pair of locations, and a flow (weight) specified for each pair of facilities (e.g., the amount of supplies transported between the two facilities), assigning all 30 facilities to the 30 different locations with the goal of minimizing the sum of the distances multiplied by the corresponding flows.

Despite its apparent simplicity, the problem is NP-Hard and the number of possible assignments is extremely large, so that even if you could check a trillion assignments per second, this process would take over 100 times the age of the universe. However, once the algorithms and software necessary to tackle the previously unsolved problem on a computational grid were developed, solving the problem required nearly a week, with a computational endeavor involving more than 1,000 computational resources working simultaneously at eight institutions geographically distributed in different parts of the world.

The FightAIDS@Home project, which is based on the volunteered computing power of the World Community Grid, aims at testing candidate compounds against the variations (or "mutants") of HIV that can arise and cause drug resistance.

During November 2009, the project identified several fragments as new candidates for a novel binding site on the peripheral surface of HIV protease. These fragments docked well against the "exo site" and in vitro studies (i.e., "wet lab" experiments in test tubes) will assess their potencies. If these wet lab experiments produce promising results, then these fragments could form the foundation for the development of "allosteric inhibitors" of HIV protease (i.e., "flexibility wedges" that can disrupt the conformational changes that HIV protease must undergo in order to function). These allosteric inhibitors could represent a totally new class of anti-AIDS compounds.

These two examples clearly explain why scientists are now routinely supported in their research by grid infrastructures. But what about business and casual users? Although projects such as BEinGRID have reported some successful business experiments that may profit from execution in grid environments, it appears that there is not a general business case for the grid. However, recent advances in virtualization techniques, coupled with the increased Internet bandwidth now available led in 2007 to the concept of cloud computing. The emergence of this new paradigm is mainly based on its simplicity and the affordable price for seamless access to both computational and storage resources.

Virtualization enables cloud computing, providing the ability to run legacy applications on older operating systems, creation of a single system image starting from an heterogeneous collection of machines such as those traditionally found in grid environments, and faster job migration within different virtual machines running on the same hardware. For grid and cloud computing, virtualization is the key for provisioning and fair resource allocation. From the security point of view, since virtual machines run isolated in their sandboxes, this provides an additional protection against malicious or faulty codes.

Clouds provide access to inexpensive hardware and storage resources through very simple APIs, and are based on a pay per use model, so that renting these resources is usually much cheaper than acquiring dedicated new ones. Moreover, people is becoming comfortable with storing their data remotely in a cloud environment. Therefore, clouds are being increasingly used by scientists, small and medium sized enterprises, and casual users.

Grids, clouds and virtualization are exciting technologies that are going to become prominent in the next few years; we expect a wide proliferation in their use, especially clouds since these distributed computing facilities are already accessible at a reasonable cost to many potential users. We also expect grids and clouds to play an ever increasing role in the field of scientific research. It is therefore necessary a thorough understanding of principles and techniques of these fields, and the main aim of this book is to foster awareness of the essential ideas by exploring current and future developments in the area.

The idea of writing this book dates back to the highly successful Grids, Clouds and Virtualization Workshop that we organized in conjunction with the 4th International Conference on Grid and Pervasive Computing (GPC 2009) held in Geneva, 4-8 May 2009. We were contacted by Mr. Wayne Wheeler of Springer and, after an insightful discussion, we agreed to serve as the editors for the book. Indeed, it is virtually impossible for a single person to write a book covering all of the important aspects of grids, clouds and virtualization while maintaining the required depth, consistency and appeal.

We invited many well known and internationally recognized experts, asking them to contribute their expertise. The book delves into details of grids, clouds and virtualization, guiding the reader through a collection of chapters dealing with key topics. The bibliography rather than being exhaustive, covers essential reference material. The aim is to avoid an encyclopedic approach since we believe that an attempt to cover everything will instead fail to convey any useful information to the

interested readers, an audience including researchers actively involved in the field, undergraduate and graduate students, system designers and programmers, and IT policy makers.

The book may serve both as an introduction and as a technical reference. Our desire and hope is that it will be useful to many people familiarizing with the subject and will contribute to new advances in the field.

Lecce, Italy *Massimo Cafaro*
April 2010 *Giovanni Aloisio*

Acknowledgements

Every book requires months of preparations, and this book is no exception. We would like to express our gratitude to the contributors for their participation in this project. Without their technical expertise, patience and efforts, this book would not have been possible.

We are also indebted with the Springer editorial team for their cooperation efforts, that made this book a reality. In particular, we are deeply grateful to Mr. Wayne Wheeler, Senior Editor, for his initial proposal and continuous encouragements. We serve as the editors of this book owing to his incredible skills and energy. Special thanks must also go to Mr. Simon Rees, Senior Editorial Assistant, for his dedication, support and punctuality.

Contents

List of Contributors

Sándor Ács
Laboratory of Parallel and Distributed Systems, MTA SZTAKI
e-mail: acs@sztaki.hu

Giovanni Aloisio
University of Salento, Lecce, Italy
e-mail: giovanni.aloisio@unisalento.it

Massimo Cafaro
University of Salento, Lecce, Italy
e-mail: massimo.cafaro@unisalento.it

Ewa Deelman
University of Southern California, Marina del Rey, CA
e-mail: deelman@isi.edu

Marcos Dias de Assunção
INRIA - LIP Laboratory, University of Lyon, France
e-mail: marcos.dias.de.assuncao@ens-lyon.fr

Zoltan Farkas
Laboratory of Parallel and Distributed Systems, MTA SZTAKI
e-mail: zfarkas@sztaki.hu

Eduardo Huedo
Universidad Complutense de Madrid, 28040 Madrid
e-mail: ehuedo@fdi.ucm.es

Gideon Juve
University of Southern California, Marina del Rey, CA
e-mail: juve@usc.edu

Peter Kacsuk
Laboratory of Parallel and Distributed Systems, MTA SZTAKI

e-mail: kacsuk@sztaki.hu

Miklos Kozlovszky
Laboratory of Parallel and Distributed Systems, MTA SZTAKI
e-mail: m.kozlovszky@sztaki.hu

J. Lakshmi
SERC, Indian Institute of Science, Bangalore-560012, India
e-mail: jlakshmi@serc.iisc.ernet.in

Laurent Lefèvre
INRIA - LIP Laboratory, University of Lyon, France
e-mail: laurent.lefevre@inria.fr

Ignacio M. Llorente
Universidad Complutense de Madrid, 28040 Madrid
e-mail: llorente@dacya.ucm.es

Attila Csaba Marosi
Laboratory of Parallel and Distributed Systems, MTA SZTAKI e-mail: atisu@sztaki.hu

Rubén S. Montero
Universidad Complutense de Madrid, 28040 Madrid
e-mail: rubensm@dacya.ucm.es

S. K. Nandy
SERC, Indian Institute of Science, Bangalore-560012, India
e-mail: nandy@serc.iisc.ernet.in

Anne-Cécile Orgerie
ENS Lyon - LIP Laboratory, University of Lyon, France
e-mail: annececile.orgerie@ens-lyon.fr

Rafael Moreno-Vozmediano
Universidad Complutense de Madrid, 28040 Madrid
e-mail: rmoreno@dacya.ucm.es

Grids, Clouds and Virtualization

Massimo Cafaro and Giovanni Aloisio

Abstract This chapter introduces and puts in context Grids, Clouds and Virtualization. Grids promised to deliver computing power on demand. However, despite a decade of active research, no viable commercial grid computing provider has emerged. On the other hand, it is widely believed - especially in the Business World - that HPC will eventually become a commodity. Just as some commercial consumers of electricity have mission requirements that necessitate they generate their own power, some consumers of computational resources will continue to need to provision their own supercomputers. Clouds are a recent business-oriented development with the potential to render this eventually as rare as organizations that generate their own electricity today, even among institutions who currently consider themselves the unassailable elite of the HPC business. Finally, Virtualization is one of the key technologies enabling many different Clouds. We begin with a brief history in order to put them in context, and recall the basic principles and concepts underlying and clearly differentiating them. A thorough overview and survey of existing technologies provides the basis to delve into details as the reader progresses through the book.

1 Introduction

This chapter introduces and puts in context Grids, Clouds and Virtualization [17]. Grids promised to deliver computing power on demand. However, despite a decade of active research, no viable commercial grid computing provider has emerged. On the other hand, it is widely believed - especially in the Business World - that HPC

Massimo Cafaro
University of Salento, Italy, e-mail: massimo.cafaro@unisalento.it

Giovanni Aloisio
University of Salento, Italy, e-mail: giovanni.aloisio@unisalento.it

will eventually become a commodity. Just as some commercial consumers of electricity have mission requirements that necessitate they generate their own power, some consumers of computational resources will continue to need to provision their own supercomputers. Clouds are a recent business-oriented development with the potential to render this eventually as rare as organizations that generate their own electricity today, even among institutions who currently consider themselves the unassailable elite of the HPC business. Finally, Virtualization is one of the key technologies enabling many different Clouds. We begin with a brief history in order to put them in context, and recall the basic principles and concepts underlying and clearly differentiating them. A thorough overview and survey of existing technologies and projects provides the basis to delve into details as the reader progresses through the book.

2 A bit of history

The history of Grids and Clouds may be traced back to the 1961 MIT Centennial, when John McCarthy, a pioneer in mathematical theory of computation, artificial intelligence, and inventor of the Lisp programming language, first exposed the idea of utility computing: "... If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry."

Indeed, owing to the huge costs and complexity of provisioning and maintaining a data center, in the next two decades many large organizations (primarily banks) rented computing power and storage provided by mainframe computers geographically spread in the data centers of IBM and other providers. Meanwhile, mini, micro and personal computers appeared on the market. During early 1980s, the majority of the organizations acquired affordable personal computers and workstations. This was perceived as the end of utility computing until the next decade.

In 1992, Charlie Catlett and Larry Smarr introduced the concept of in their seminal paper [50]. The term metacomputing refers to computation on a virtual supercomputer assembled connecting together different resources like parallel supercomputers, data archives, storage systems, advanced visualization devices and scientific instruments using high speed net-works that link together these geographically distributed resources. The main reason for doing so is because it enables new classes of applications [14] [39] previously impossible and because it is a cost effective approach to high performance computing. Metacomputing proved to be feasible in several experiments and testbeds, including the I-WAY experiment [19] [25] and in the Globus Gusto testbed [26].

The new applications were initially classified as follows:

- desktop supercomputing;
- smart instruments;
- collaborative environments;

- distributed supercomputing.

Desktop supercomputing included applications coupling high-end graphics capabilities with remote supercomputers and/or databases; smart instruments are scientific instruments like microscopes, telescopes and satellites requiring supercomputing power to process the data produced in near real time. In the class of collaborative environments there were applications in which users at diferent locations could interact together working on a supercomputer simulation; distributed supercomputing finally was the class of application requiring multiple supercomputers to solve problems otherwise too large or whose execution was divided on different components that could benefit from execution on different architectures.

The challenges to be faced before metacomputing could be really exploited were identified as related to the following issues:

- scaling and selection;
- unpredictable structure;
- heterogeneity;
- dynamic behavior;
- multiple administrative domains.

Interestingly (form a research perspective), these are still relevant today. is a concern, because we expect that grid and cloud environments in the future will become even larger, and resources will be selected and acquired on the basis of criteria such as connectivity, cost, security and reliability. These resources will show different levels of heterogeneity, ranging from physical devices to system software and schedulers policies; moreover, traditional high performance applications are developed for a single supercomputer whose features are known a priori, e.g. the latency of the interconnection network, in contrast grid and cloud applications will run in a wide range of environments thus making impossible to predict the structure of the computation. Another concern is related to the dynamic behavior of the computation [94], since we cannot, in general, be assured that all of the system characteristics stay the same during the course of computation, e.g. the network bandwidth and latency can widely change, and there is the possibility of both network and resource failure. Finally, since the computation will usually span resources geographically spread at multiple administrative domains, there is not a single authority in charge of the system, so that different scheduling policies and authorization mechanisms must be taken into account.

In the same years, a project called Legion [33] promoted the grid object model. Legion was designed on the basis of common abstractions of the object oriented model: everything in Legion was an object with a well defined set of access method, including files, computing resources, storage etc. The basic idea was to expose the grid as a single, huge virtual machine with the aim of hiding the underlying complexity to the user. The middleware proved to be useful in several experiments, including a distributed run of an ocean model and task farmed computational chemistry simulations. However, it became immediately apparent that the majority of the people in academia were not fond of the grid object model; consequently, the attention shifted towards the use of the Globus Toolkit, which a few years later provided

an alternative software stack and became quickly the de facto standard for grid computing.

The Globus Toolkit, initially presented as a metacomputing environment [24], was one of the first middleware solutions really designed to tackle the issues related to large scale distributed computing, and its usefulness in the context of metacomputing was demonstrated in the Gusto testbed; among the distributed simulations that have been run using Globus there was SF-Express [16], the largest computer simulation of a military battle involving at the time more that 100,000 entities. Even the EuropeanData Grid project, in charge of building a European middleware for grid computing, did so leveraging many of the Globus components. This efforts led to the gLite middleware [40] which is going to be used for the analysis of experimental results of the Large Hadron Collider, the particle accelerator at CERN that recently started operations in Geneva. The middleware was also actively tested in the context of the EGEE project [41], which provided the world largest computational grid testbed to date. Another European project developed a different middleware, Unicore [21], targeting mainly HPC resources. A similar endeavor, devoted to the implementation of grid middleware for high performance computing, took place in Japan starting in 2003 with the National Research Grid Initiative (NAREGI) [43] and culminating in 2008 with the release of the middleware.

There was a pressure to grid-enable many existing, legacy products. Among the schedulers, we recall here Condor [51], Nimrod [10], Sun Grid Engine [30] and Platform Computing LSF [55]. The Condor project begun in 1988 and is therefore one of the earliest cycle scavenging software available. Designed for loosely coupled jobs, and to cope with failures, it was ported to the grid through a Globus extension aptly named Condor-G. The Nimrod system was designed to manage large parameter sweep jobs, in which the same executable was run each time with a different input. Since the jobs are independent, a grid is clearly a good fit for this situation. The Nimrod-G system, extended once again through the Globus Toolkit, was also one of the earliest projects in which the concepts of grid economy appeared. In addition to several other criteria, the user could also exploit the cost of the whole simulation when submitting a job. The system was able to charge the CPU cycles distinguishing HPC resources from traditional, off-the-shelf ones. Sun Grid Engine, an open source project led by Sun, started in 2000. The software was originally based on the Gridware Codine (COmputing in DIstributed Network Environments) scheduler. It was grid-enabled and is now being cloud-enabled. Finally, LSF was one the first commercial products offering support for grid environments through its extension named LSF MultiCluster.

The business and commercial world recognized the impact and the potential of grid computing, whose usefulness was demonstrated in hundreds of projects. However, almost all of the projects were driven by people involved in academia and targeting mainly scientific applications. Very few projects, notably the BEinGrid [20] one, were in charge of demonstrating the use of grid computing for business oriented goals. BEinGrid, with its sample of 25 business experiments was also successful in the implementation and deployment of Grid solutions in industrial key sectors. Nonetheless, grids are still not appealing to the business world, and this

is reflected in the lack of current commercial grid solutions. Platform Computing provided years ago a version of the Globus Toolkit which is now unsupported and not for sale. Legion was initially sold by Applied Meta, then by Avaki Corporation which was acquired in 2005 by Sybase, Inc. What remains of the software is now used by the company's customers to share enterprise data. Entropia, Inc., a company founded in 1997, sold distributed computing software for CPU scavenging until 2004.

The idea of harnessing idle CPUs worldwide was first exploited by the SETI@home project [11], launched in 1996. Soon a number of related projects including GIMPS [44], FightAIDS@Home [47], Folding@home [13] arised. GIMPS, the Great Internet Mersenne Prime Search, was started in January 1996 to discover new world-record-size Mersenne primes. A Mersenne prime is a prime of the form $2^P - 1$. The first Mersenne primes are 3, 7, 31, 127 (corresponding to $P = 2, 3, 5, 7$). There are only 47 known Mersenne primes, and GIMPS has found 13 of the 47 Mersenne primes ever found during its 13 year history. FightAIDS@Home uses distributed computing exploiting idle resources to accelerate research into new drug therapies for HIV, the virus that causes AIDS; FightAIDS@Home made history in September 2000 when it became the first biomedical Internet-based grid computing project. Proteins, in order to carry out their function, must take on a particular shape, also known as a fold. One of the Folding@home goals is to simulate protein folding in order to understand how proteins fold so quickly and reliably, and to learn about what happens when this process goes awry (when proteins misfold).

The initial release of the Apple Xgrid [36] [35] technology was also designed for independent, embarrassingly parallel jobs, and was later extended to support tightly coupled parallel MPI jobs. A key requirement of Xgrid was simplicity: everyone is able to setup and use an hoc grid using Xgrid, not just scientists. However, only Apple resources running the Mac OS X operating system may belong to this grid, although third-party software (an Xgrid linux agent and a Java based one) not supported or endorsed by Apple allows deploying heterogeneous grids.

3 Grids

It is interesting to begin by reviewing some of most important definitions of given during the course of research and development of the field. The earliest definition given in 1998 by I. Foster and C. Kesselman [37] is focused around on-demand access to computing, data, and services: a computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. Two years later, the definition was changed to reflect the fact that grid computing is concerned with coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. In the latest definitions of I. Foster [27] [23], a grid is described respectively as an infrastructure for and problem solving in dynamic, multi-institutional and as a system that coordinates resources that are not subject to centralized control, using

standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service. We will argue later, when comparing grids and clouds, that delivering nontrivial qualities of service is very difficult using a distributed approach instead of a centralized one.

Grids are certainly an important computing paradigm for science, and there are many different scientific communities facing large-scale problems simply too big to be faced even on a single, powerful parallel supercomputer. After more than a decade of active research and development in the field, all of the people expected grids to become a commercial service; however, this has not happened yet. Therefore, we ask ourselves: is there a general business case for grids?

To answer this question, we recall that it is now clear that grid computing benefits specific classes of applications, and that the technology itself, while powerful is probably not yet simple enough to be released commercially. This is reflected in the following sentence of Frank Gillette, Forester: "None of us have figured out a simple way to talk about (grid) . . . because it isn't simple". And, when something takes an hour or more to be explained, it certainly can't be sold easily. Therefore, application providers may prefer to integrate the required middleware into their applications, or take advantage of existing Platform and DataSynapse solutions. The issue raised by the complexity of grids proves to be a formidable barrier that we must overcome if we want grids to become a fact of life that everyone uses and nobody notices. This has been recently remarked in a position paper by G. Fox and M. Pierce [29] in which the authors discuss the fact that "Grids meet Too much Computing, Too much Data and never Too much Simplicity".

On the other hand, early testbeds and recent grids deployment have clearly shown the potential of grid computing. In particular, loosely coupled parallel applications, parameter sweep studies and applications described by a workflow assembling and orchestrating several different components are viable candidates for grid computing. Example applications include those from the High Energy Phisics (HEP) community, falling in the parameter sweep class, and those from the bioinformatics community, falling in the workflow class. Typical loosely coupled parallel applications are exemplified by climate simulations, in which the initial functional decomposition of the problem is well suited for execution on grids linking together distributed supercomputers; the blocks identified by the initial decomposition (atmosphere, land, sea, ice etc) are then run on those supercomputers and exchange data infrequently.

When the vendors realized that this kind of applications were far beyond the common needs of the majority of the people, the attention shifted to the emerging field of cloud computing. Here, in contrast to grid computing, the emphasis was put on simplicity as the ingredient to make clouds easily accessible.

4 Clouds

It is rather difficult to precisely define what clouds and are, especially taking into account the many possible different uses. We recall here that the same also happened

in the context of grids. The concept as we all know it today was introduced in late 2007 [53]. Among the many definitions that were given since then, we report the following ones:

- Gartner: Cloud computing is a style of computing where massively scalable IT-related capabilities are provided 'as a service' across the Internet to multiple external customers;
- Forrester: A pool of abstracted, highly scalable, and managed infrastructure capable of hosting end-customer applications and billed by consumption;
- IBM: An emerging computing paradigm where data and services reside in massively scalable data centers and can be ubiquitously accessed from any connected devices over the internet.

How clouds are actually perceived by the people appears to be much broader in scope. The following five definitions [48] expose several perspectives on this subject.

1. Cloud computing refers (for many) to a variety of services available over the Internet that deliver compute functionality on the service provider's infrastructure (e.g. Google Apps or Amazon EC2 or Salesforce.com). A cloud computing environment may actually be hosted on either a grid or utility computing environment, but that doesn't matter to a service user;
2. - Cloud computing = Grid computing. The workload is sent to the IT infrastructure that consists of dispatching masters and working slave nodes. The masters control resource distributions to the workload (how many slaves run the parallelized workload). This is transparent to the client, who only sees that workload has been dispatched to the cloud/grid and results are returned to it. The slaves may or may not be virtual hosts;
   - Cloud computing = . This is the Google apps model, where apps are located 'in the cloud', i.e. somewhere in the Web;
   - Cloud computing = . This is the Amazon EC2 et al model where an external entity maintains the IT infrastructure (masters/slaves) and the client buys time/resources on this infrastructure. This is 'in the cloud' in so much that it is across the Web, outside of the organization that is leasing time off it;
3. The cloud simply refers to the move from local to service on the Web. From storing files locally to storing them in secure scalable environments. From doing apps that are limited to GB spaces to now apps that have no upper boundary, from using Microsoft Office to using a Web-based office. Somewhere in 2005-2008 storage online got cheaper and more secure than storing locally or on your own server. This is the cloud. It encompasses grid computing, larger databases like Bigtable, caching, always accessible, failover, redundant, scalable, and all sorts of things. Think of it as a further move into the Internet. It also has large implications for such battles as static vs. dynamic, RDBMS vs. BigTable and flat data views. The whole structure of business that relies on IT infrastructure will change, programmers will drive the cloud and there will be lots of rich programmers at the end. It is like the move from mainframe to personal computers. Now you have a personal space in the clouds;

4. Grid and Cloud are not exclusive of each other . . . Our customers view it this way: Cloud is pay for usage (i.e. you don't necessarily own the resources) Grid is how to schedule the work - regardless where you run it You can use a cloud without a grid, a grid without a cloud. Or you can use a grid on a cloud;
5. I typically break up the idea of cloud computing into three camps:

   - Enablers - These are companies that enable the underlying infrastructures or the basic building blocks. These companies are typically focused on data center automation and or server virtualization (VMware/EMC,Citrix,BladeLogic, RedHat, Intel, Sun, IBM, Enomalism, etc.);
   - Providers - (Amazon Web Services, Rackspace, Google, Microsoft). The ones with the budgets and know-how to build out global computing environments costing millions or even billions of dollars. Cloud providers typically offer their infrastructure or platform. Frequently these As a Service offerings are billed and consumed on a utility basis;
   - Consumers - On the other side of the spectrum I see the consumers companies that build or improve their Web applications on top of existing clouds of computing capacity without the need to invest in data centres or any physical infrastructure. Often these two groups can be one in the same such as Amazon (SQS,SDB,etc), Google (Apps) and Salesforce (Force). But they can also be new startups that provide tools and services that sit on top of the cloud (Cloud management);

   Cloud consumers can be a fairly broad group including just about any application that is provided via a Web-based service like a Webmail, blogs, social network, etc. Cloud computing from the consumer point of view is becoming the only way you build, host and deploy a scalable Web application.

On the other hand the main findings of the Cloud BoF held at OGF22, Cambridge, MA, on 27 Feb 2008 were the following ones:

- Clouds are "Virtual Clusters" ("Virtual Grids") of possibly "Virtual Machines";
- They may cross administrative domains or may "just be a single cluster"; the user cannot and does not want to know;
- Clouds support access (lease of) computer instances;
- Instances accept data and job descriptions (code) and return results that are data and status flags;
- Each Cloud is a "Narrow" (perhaps internally proprietary) Grid;
- When does Cloud concept work:
  - Parameter searches, LHC style data analysis . . .
  - Common case (most likely success case for clouds) versus corner case?
- Clouds can be built from Grids;
- Grids can be built from Clouds;
- Geoffrey Fox: difficult to compare grids and clouds because neither term has an agreed definition;
- Unlike grids, clouds expose a simple, high-level interface;

- There are numerous technical issues:
  - performance overhead, cost, security, computing model, data-compute affinity, schedulers and QoS, link clouds (e.g. compute-data), . . . ;
  - What happens when a cloud goes down? What about interoperability of clouds? Standardization? Is it just another service?

A recent report [12] present an in-depth view of cloud computing. In what follows we will try to make things easier to understand, summarizing our perspective. In order to do so, we begin discussing the main features and characteristics of clouds as currently deployed and made available to the people. The main characteristic of cloud computing certainly is its focus on virtualization. Clouds are succeeding owing to the fact that the underlying infrastructure and physical location are fully transparent to the user.

Beyond that, clouds also exhibit excellent scalability allowing users to run increasingly complex applications and breaking the overall workload into manageable pieces served by the easily expandable cloud infrastructure. This flexibility is a key ingredient and it is very appealing to the users. Clouds can adapt dynamically to both consumer and commercial workloads, providing efficient access through a Service Oriented Architecture to a computing infrastructure delivering dynamic provisioning of shared compute resources.

An attempt to provide a comprehensive comparison of grids and clouds is of course available [28]. However, we now compare and contrast grids and clouds, in order to highlight what we think are key differences. Grids are based on open standards; the standardization process happens in organizations such as the , etc. Clouds, in contrast, do not provide standardized interfaces. Especially commercial clouds solutions are based on proprietary protocols, which are not disclosed to the scientific community. A related aspect is that of . While in grid environments interoperability ha become increasingly important, and many efforts are devoted to this topic [15], clouds are not interoperable and will not be in the short-term period. Vendors have no interest at the moment to provide interoperability among their cloud infrastructures.

Almost all of the currently deployed grids have been publicly funded and operated. This is of course a very slow process, when compared to clouds which are instead privately funded and operated. Many companies have already invested and continue to invest a huge amount of money to develop cloud technologies; however, considering the limited amount of funding available to scientists, we must remark the excellent results in the field of grid computing.

Examining how grids are operated, it is easy to see that, since the beginning, there was a design requirement to build grid infrastructure tying together distributed administrative domains, possibly geographically spread. On the contrary, clouds are managed by a single company/entity/administrative domain. Everything is centralized, the infrastructure is hosted in a huge centre and only the clients are actually geographically distributed. The architecture is basically .

We note here that, despite its simplicity, the client-server architecture is still the most widely used in the commercial world, owing to the fact that it is very hard to

beat hosted/managed services with regard to performance and resiliency: these services are currently geographically replicated and hugely provisioned and can guarantee/meet a specific . Highly distributed architectures, including systems, while in principle are theoretically more resistant to threats such as Denial of Service attacks etc, still do not provide the same performance guarantee. For instance, a P2P Google service is deemed to be impossible [42] (although Google uses internally their own P2P system, based on the concept of MapReduce [18]), and people who do it for business, today don't do it peer-to-peer, with the notable exception of the Skype service [49].

Coming to the main purpose of grid and cloud systems, it is immediately evident that for grid systems the main raison d'etre is resource sharing. Cloud systems are instead intended to provide seamless access access to a huge, scalable infrastructure. Given the different purpose they serve, it comes to no surprise that these systems target different classes of applications. As we have already noted, grids are well suited for scientific research, and for the needs of high-end users. Clouds are mainly used today for data analysis, information processing and data mining.

We conclude this section recalling that Many Task Computing (MTC) and High Throughput Computing (HTC) service providers and resource service providers can benefit from the economies of scale that clouds can deliver [52], making this kind of infrastructure appealing to the different stakeholders involved in.

5 Virtualization

The (VM) concept [45] dates back to the 60s; it was introduced by IBM as a mean to provide concurrent, interactive access to their mainframe computers. A VM was an instance of the physical machine and gave users the illusion of accessing the physical machine directly. VMs were used to transparently enable time-sharing and resource-sharing on the (at the time) highly expensive hardware. Each VM was a fully protected and isolated copy of the underlying system. Virtualization was thus used to reduce the hardware costs on one side and to improve the overall productivity by letting many more users work on it simultaneously. However, during the course of years, the hardware got cheaper and simultaneously multiprocessing operating systems emerged. As a consequence, VMs were almost extinct in 1970s and 1980s, but the emergence of wide varieties of PC based hardware and operating systems in 1990s, revived virtualization ideas.

Virtualization technologies represent a key enabler of cloud computing, along with the recent advent of web 2.0 and the increased bandwidth availability on the Internet. The most prominent feature is the ability to install multiple OS on different virtual machines on the same physical machine. In turn, this provides the additional benefits of overall cost reduction owing to the use of less hardware and consequently less power. As a useful side effect, we also note here that virtualization generally leads to increased machine utilization. The main aim of virtualization technologies is to hide the physical characteristics of computing resources from the way in which

other systems, applications, or end users interact with those resources. In this book, Lakshmi et al. [38] propose an end-to-end system virtualization architecture and thoroughly analyze it.

Among the many benefits provided by virtualization, we recall the ability to run legacy applications requiring an older platform and/or OS, the possibility of creating a single system image starting from an heterogeneous collection of machines such as those traditionally found in grid environments, and faster job migration within different virtual machines running on the same hardware. For grid and cloud computing, virtualization is the key for provisioning and fair resource allocation. From the security point of view, since virtual machines run isolated in their sandboxes, this provides an additional protection against malicious or faulty codes.

Besides computing, storage may be virtualized too. Through the aggregation of multiple smaller storage devices characterized by attributes such as performance, availability, capacity and cost/capacity, it becomes possible to present them as one or more virtual storage devices exhibiting better performance, availability, capacity and cost/capacity properties. In turn, this clearly enhances the overall manageability of storage and provides better sharing of storage resources.

A virtualization layer provides the required infrastructural support exploiting lower-level hardware resources in order to create multiple independent virtual machines that are isolated from each other. This layer, traditionally called (VMM), usually sits on top of the hardware and below the operating system. Virtualization as an abstraction, can take place at several different levels, including instruction set level, (HAL), OS level (system call interface), user-level library interface, and the application level.

Virtualizing the instruction set requires emulation of the instruction set, i.e. interpreting the instructions in software. A notable example is Rosetta [9], which is a lightweight dynamic translator for Mac OS X distributed by Apple. Its purpose is to enable legacy applications compiled for the PowerPC family of processors to run on current Apple systems that use Intel processors. Other examples include Bochs [1], QEMU [2] and BIRD [46].

HAL level virtual machines are based on abstractions lying between a real machine and an emulator; in this case a virtual machine is an environment created and managed by a VMM. While an emulator's functionalities provide a complete layer between the operating system or applications and the hardware, a VMM is in charge of managing one or more VMs and each VM in turn provides facilities to an OS or application as if it is run in a normal environment, directly on the hardware. Examples include VMware [3], Denali [54], Xen [4], Parallels [5] and Plex86[6].

Abstracting virtualization at the OS level requires providing a virtualized system call interface; this involves working on top of the OS or at least as a OS module. Owing to the fact that system calls are the only way of communication from user-space processes to kernel-space, the virtualization software can control user-space processes by managing this system interface. Additionally, besides system's library, usually applications also link code provided by third-party user-level libraries. Virtualizing this library interface can be done by controlling the communication link between the applications and the rest of the system through well defined API hooks.

Therefore, it is possible to expose a different implementation using the same set of APIs. As an example, WINE HQ [7] and CrossOver [8] support running Windows applications respectively on Unix and Mac OS X.

Virtualization at the application is not based on the insertion of a virtualization layer in the middle. Instead, this kind of abstraction is implemented as an application that eventually creates a virtual machine. The created VM could range in complexity from a simple language interpreter to a very complex Java Virtual machine (JVM).

All of these virtualization technologies, are based on approaches differing significantly when evaluated with regard to several metrics such as performance, flexibility, simplicity, resource consumption, and scalability. Therefore, it is important to understand their usage scenarios as well. Instruction set emulators are often characterized by very high latencies which makes them impractical to use on a regular basis (with the notable exception of Rosetta). Their main goal is to provide an environment especially tailored to debugging and learning purposes, since every component is implemented in software and is fully under user's control.

Commercial virtual machines operating at HAL level, besides offering extremely low latencies, also give the flexibility of using different OSes or different versions of the same OS on the same machine; these VMs present a complete machine interface, but of course this demands a much higher amount of resources. OS level virtual machines are useful in those cases in which this kind of flexibility is not required, because resource requirements are much lower, performance better, and manipulations (e.g. creation) faster. It is worth noting here that the attained level of isolation is lower, since all of the VMs use the same kernel and can potentially affect the whole system.

Finally, library-level virtualization technologies prove to be extremely lightweight. On the contrary, Application- level virtualization suffers from extra overhead being in the application-level. The latter technology is increasingly applied in mobile computing and as a building block for trusted computing infrastructures.

6 Technologies

In this Section, we review the most important technologies available. We recall here the following:

- Amazon Elastic Compute Cloud (EC2);
- Google App Engine;
- Hadoop;
- HP Labs Enterprise Cloud Assure and Cloud Software Platform (Cirious);
- IBM Smart Business cloud solutions;
- Sun Cloud;
- Oracle On Demand and Platform for SaaS;
- SGI Cyclone;
- Microsoft Azure.

The Amazon cloud is probably the most used cloud computing environment. It works as follows. You start by creating an Amazon Machine Image (AMI) containing all your software, including your operating system and associated configuration settings, applications, libraries, etc. Amazon provides all of the necessary tools to create and package your AMI. Then, you upload this AMI to the cloud through the Amazon S3 (Amazon Simple Storage Service) service. This gives us reliable, secure access to your AMI. In order to run your applications, you need to register your AMI with Amazon EC2. This step allows verifying that your AMI has been uploaded correctly and uniquely identifies it with an identifier called AMI ID. Using the AMI ID and the Amazon EC2 web service APIs it is easy to run, monitor, and terminate as many instances of the AMI as required. Amazon provides command line tools and Java libraries, and you may also take advantage of SOAP or Query based APIs. Amazon also lets you launch pre-configured AMIs provided by Amazon or shared by another user While AMI instances are running, you are billed for the computing and network resources that they consume.

Among the applications that have actually been run on EC2, we recall here web hosting, parallel processing, graphics rendering, financial modeling, web crawling and genomics analysis. To be fair, while parallel applications cab be run on the Amazon cloud, performances are of course far from the current result you can achieve on a dedicated supercomputing infrastructure, owing to less powerful CPUs and basic interconnection networks with very high latency and low bandwidth when compared to the high-end interconnects traditionally available on supercomputers. For instance, a recent work [22] is related to deploying and running a coupled climate simulation on EC2. The results obtained are comparable to running the simulation on a low-cost beowulf cluster assembled using commodity, off-the-shelf hardware.

The Google App Engine exploits the underlying Google infrastructure, the same that was built to bring access to their web search engine and Gmail to people worldwide. Leveraging a powerful network of data centers, this infrastructure is aimed at scalable web applications built using Python and, more recently, Java. Developers are assigned a free quota of 500 MB of storage and enough CPU and network bandwidth to sustain around 5 million page views per month for a typical app; it is possible to purchase additional storage and bandwidth to go beyond the limits. It is worth noting here that Google provides proprietary APIs to take advantage of the infrastructure, so that writing an application is considerably easier, but at the same time once a development team selects Google App Engine as the environment of choice for their web application, they remain locked to the platform, because a move will require significant code rewrites. The competing Amazon EC2 service is different, in that it allows developers to create virtual server instances, and leaves them the option of moving their applications easily to other machines.

Hadoop [34] is an open-source Apache Software Foundation project sponsored by Yahoo! whose intent is to reproduce the proprietary software infrastructure developed by Google. It provides a parallel programming model, [18], heavily exploited by Google, a distributed file system, a parallel database, a data collection system for managing large distributed systems, a data warehouse infrastructure that provides data summarization and ad hoc querying, a high-level data-flow language and ex-

ecution framework for parallel computation and a high-performance coordination service for distributed applications.

The software is widely used by many companies and researchers. As an example of usage, we recall here the New York Times task of generating about 11 millions of PDF files related to the public domain articles from 1851-1922 by gluing together multiple TIFF images per article [31] . Derek Gottfried uploaded 4 TB of data to the Amazon EC2 infrastructure using the S3 service, wrote the software to pull all the parts that make up an article out of S3, generate a PDF from them and store the PDF back in S3 using open source Java components, and finally he deployed Hadoop on a cluster of of EC2 machines. Overall, the whole task took less than 24 hours using 100 instances on Amazon EC2, and generated additional 1.5 TB of data to be stored in S3.

Regarding the MapReduce programming model, it is a data-flow model between services where services can do useful document-oriented data parallel applications including reductions; using Hadoop the decomposition of services onto cluster in a clouds is fully automated. The key idea is inspired by some primitives of the LISP programming language. Both the input and the output are key/value pairs, and developers needs to implement two functions, map and reduce:

- $map(in\_key, in\_value) \rightarrow list(out\_key, intermediate\_value)$;
- $reduce(out\_key, list(intermediate\_value)) \rightarrow list(out\_value)$;

The former processes an input key/value pair, producing a set of intermediate pairs. The latter is in charge of combining all of the intermediate values related to a particular key, outputting a set of merged output values (usually just one). MapReduce is often explained illustrating a possible solution to the problem of counting the number of occurrences of each word in a large collection of documents. The following pseudocode refers to the functions that needs to be implemented.

```
map(String input_key, String input_value):
    // input_key: document name
    // input_value: document contents
    for each word w in input_value:
      EmitIntermediate(w, "1");


 reduce(String output_key, Iterator intermediate_values):
    // output_key: a word
    // output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
      result += ParseInt(v);
    Emit(AsString(result));
```

The map function emits in output each word together with an associated count of occurrences (in this simple example just one). The reduce function provides the required result by summing all of the counts emitted for a specific word. MapReduce

implementations (Google App Engine and Hadoop for instacne) then automatically parallelize and execute the program on a large cluster of commodity machines. The runtime system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing required inter-machine communication. While MapReduce certainly limits the scope of parallel applications that can be developed to a particular type of interaction, it allows programmers with no previous experience with parallel and distributed computing to easily utilize the resources of a large distributed system. A typical MapReduce computation processes many terabytes of data on hundreds or thousands of machines. Programmers find the the model easy to use (and it certainly is when compared to more advanced interactions among tasks), and more than 100K MapReduce jobs are routinely executed on Google's and Amazon's clusters every day.

7 The economics

Grid Computing leverages the techniques of clustering where multiple independent clusters act like a grid due to their nature of not being located in a single domain. This leads to the : because of the distributed nature of the Grid the computational nodes could be located anywhere in the world. It is fine having all that CPU power available, but the data on which the CPU performs its operations could be thousands of miles away, causing a delay (latency) between data fetch and execution. CPUs need to be fed with different volumes of data depending on the tasks they are processing. Running a data intensive process with disparate data sources can create an I/O bottleneck causing the CPU to run inefficiently, and affecting economic viability. This also applies to clouds but, owing to the centralized nature of cloud environments and to the requirements of cloud applications which usually move a relatively scarce volume of data (with some exceptions), the problem is far less critical. Storage management, security provisioning and data movement are the nuts to be cracked in order for grids to succeed; more important than these technical limitations, is becoming the lack of business buy in.

The nature of Grid/Cloud computing means a business has to migrate its application and data to a third-party solution. This creates huge barriers to the uptake, at least initially. Financial business institutions need to know that grid vendors understand their business, not just the portfolio of applications they ran and the infrastructure they ran upon. This is critical to them. They need to know that whoever supports their systems knows exactly what the effect of any change could potentially make to their shareholders. The other bridge that has to be crossed is that of data security and confidentiality. For many businesses their data is the most sensitive, business critical thing they possess. To hand this over to a third-party is simply not going to happen. Banks are happy to outsource part of their services, but want to be in control of the hardware and software - basically using the outsourcer as an agency for staff.

Jim Gray (Turing Award in 1998, lost at sea in 2007) published in 2003 an interesting paper on Distributed Computing Economics [32]. As stated in this 2003 paper, "Computing economics are changing. Today there is rough price parity between (1) one database access, (2) ten bytes of network traffic, (3) 100,000 instructions, (4) 10 bytes of disk storage, and (5) a megabyte of disk bandwidth. This has implications for how one structures Internet-scale distributed computing: one puts computing as close to the data as possible in order to avoid expensive network traffic". This is exactly the data-compute affinity problem that plagues Grids.

The recurrent theme of this analysis is that On Demand computing is only economical for very cpu-intensive (100,000 instructions per byte or a cpu-day-per gigabyte of network traffic) applications. Pre-provisioned computing is likely to be more economical for most applications, especially data-intensive ones. If Internet access prices drop faster than Moore's law, the analysis fails. If instead Internet access prices drop slower than Moore's law, the analysis becomes even stronger.

What are the economic issues of moving a task from one computer to another or from one place to another? A computation task has four characteristic demands:

- Networking – delivering questions and answers,
- Computation – transforming information to produce new information,
- Database Access – access to reference information needed by the computation,
- Database Storage – long term storage of information (needed for later access).

The ratios among these quantities and their relative costs are pivotal: It is fine to send a GB over the network if it saves years of computation – but it is not economic to send a kilobyte question if the answer could be computed locally in a second.

7.1 The economics in 2003

To make the economics tangible, take the following baseline hardware parameters:

1. 2 GHz cpu with 2GB ram (cabinet and networking);
2. 200 GB disk with 100 accesses/ second and 50MB/s transfer;
3. 1 Gbps Ethernet port-pair;
4. 1 Mbps WAN link.

From this Gray concludes that one dollar equates to

- 1 GB sent over the WAN;
- 10 Tops (tera cpu operations);
- 8 hours of cpu time;
- 1 GB disk space;
- 10 M database accesses;
- 10 TB of disk bandwidth;
- 10 TB of LAN bandwidth.

Let us now think about some of the implications and consequences. Beowulf networking is 10,000x cheaper than WAN networking, and a factor of 105 matters. Moreover, the cheapest and fastest way to move a Terabyte cross country is sneakernet (the transfer of electronic information, especially computer files, by physically carrying removable media such as magnetic tape, compact discs, DVDs, USB flash drives or external drives from one computer to another): 24 hours = 4 MB/s, 50$ shipping vs 1,000$ WAN cost; A few real examples follow. Google has reportedly used a sneakernet to transport datasets too large for current computer networks, up to 120TB in size. The SETI@home project uses a sneakernet to overcome bandwidth limitations: data recorded by the radio telescope in Arecibo, Puerto Rico is stored on magnetic tapes which are then shipped to Berkeley, California for processing. In 2005, Jim Gray reported sending hard drives and even ”metal boxes with processors” to transport large amounts of data by postal mail.

The conclusions of Gray are that... ”To the extent that grid/cloud is like Seti@Home or ZetaNet or Folding@home or it is a great thing; the extent that the grid/cloud is MPI or data analysis, it fails on economic grounds: move the programs to the data, not the data to the programs; the Internet is not the cpu backplane”. This economic fact should not be hidden from the academic/scientific research community.

7.2 The economics in 2010

It is worth recalling here that when Jim Gray published his paper, the fastest Supercomputer operated at a speed of 36 TFLOPS; the fastest supercomputer available today provides computing power in excess of 1.7 PFLOPS, and a new IBM Blue Gene/Q is planned for 2010-2012 which will operate at 10 PFLOPS, practically outstripping Moore’s law by a factor of 10. Internet access prices have fallen and bandwidth has increased since 2003, but more slowly than processing power. Therefore, the economics are even worse than in 2003, and it follows that, at the moment, there are too many issues and costs with network traffic and data movements to allow Clouds to happen for all kind of customers. The majority of routine tasks, which are not processor intensive and time critical, are the most likely candidates to be migrated to cloud computing, even though these may be the least economical to be ported to that architecture. Among the many processor intensive applications, a few selected applications such as image rendering or finite modelling appears to be good candidate applications.

8 Applications

We now briefly consider what type of applications are best suited for execution on grids and clouds respectively. The reader should take into account the issues we discussed regarding the movement of both input and output data, especially regarding

large datasets. Traditional sequential batch jobs, can be run without problems on both grids and clouds.

Parallel jobs can be run without issues on a single parallel supercomputer or cluster belonging to a grid; running them on clouds require an initial effort to properly setup the environment and is likely to achieve performances similar to those on a self-made, assembled beowulf cluster using a low-cost interconnection network and off-the-shelf components.

Parameter sweep applications, in which the same executable is run each time using a different set of input files, are good candidates for execution in both grids and clouds environments: in general, loosely-coupled applications with infrequent or absent interprocess communications perform well.

Finally, complex applications with data and compute dependencies are usually described as workflow applications and characterized by a graph whose vertices represent applications and whose arcs model dependencies among the vertices. When all of the nodes in the graph represent sequential batch jobs, a workflow can be run on clouds (we already noted that parallel applications are not well suited to clouds). Conversely, if some of the nodes requires parallel processing grids are more appropriate.

9 Conclusions

This chapter introduced the concepts of grids, clouds and virtualization. We have discussed the key role of grids in advancing the state of the art in distributed computing, and how the interest for these technologies gave rise to a plethora of projects, aimed both at developing the needed middleware stack and exploiting the new paradigm to accelerate science discovery. We have seen that despite a decade of active research, no viable commercial grid computing provider has emerged, and have investigated the reasons behind. The main problems seems to be related to the underlying complexity of deploying and managing a grid, even though there are scientific applications that may actually benefit from grid computing (in particular complex applications requiring a distributed workflow model of execution). The other problem, data-compute affinity, affects both grids and clouds, even though in the cloud case the volume of data to be transferred is usually relatively scarce. We have then examined the emerging cloud computing paradigm, comparing and contrasting it with grids from many perspectives. Clouds builds on virtualization technologies which act as one of the key enablers, and are well suited to both loosely-coupled applications with infrequent or absent interprocess communications (such as parameters sweep studies) and to MapReduce based applications. Our review of virtualization/cloud technologies, puts in context cloud computing environments. Finally, we have briefly discussed the economics behind grids and clouds, laying out the foundations and providing fertile ground for the next chapters of this book.

Acknowledgements

The authors with to thank Martin Walker for the insightful discussions on grids and clouds in the context of the SEPAC grid project; part of the materials of this chapter are based on one of his many interesting presentations.

References

1. _: URL http://bochs.sourceforge.net
2. _: URL http://wiki.qemu.org
3. _: URL http://www.vmware.com
4. _: URL http://xen.org
5. _: URL http://www.parallels.com
6. _: URL http://plex86.sourceforge.net
7. _: URL http://www.winehq.org
8. _: URL http://www.codeweavers.com/products/cxmac
9. _: URL http://developer.apple.com/legacy/mac/library/documentation/MacOSX/Conceptual/\\universal_binary/universal_binary.pdf
10. Abramson, D., Buyya, R., Giddy, J.: A computational economy for grid computing and its implementation in the nimrod-g resource broker. Future Gener. Comput. Syst. **18**(8), 1061–1074 (2002). DOI http://dx.doi.org/10.1016/S0167-739X(02)00085-7
11. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: Seti@home: an experiment in public-resource computing. Commun. ACM **45**(11), 56–61 (2002). DOI http://doi.acm.org/10.1145/581571.581573
12. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley (2009). URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html
13. Beberg, A.L., Ensign, D.L., Jayachandran, G., Khaliq, S., Pande, V.S.: Folding@home: Lessons from eight years of volunteer distributed computing. In: IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, pp. 1–8. IEEE Computer Society, Washington, DC, USA (2009). DOI http://dx.doi.org/10.1109/IPDPS.2009.5160922
14. Benger, W., Foster, I.T., Novotny, J., Seidel, E., Shalf, J., Smith, W., Walker, P.: Numerical relativity in a distributed environment. In: Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing. SIAM (1999)
15. Boardman, R., Crouch, S., Mills, H., Newhouse, S., Papay, J.: Towards grid interoperability. In: All Hands Meeting 2007, OMII-UK Workshop (2007)
16. Brunett, S., Davis, D., Gottschalk, T., Messina, P., Kesselman, C.: Implementing distributed synthetic forces simulations in metacomputing environments. In: HCW '98: Proceedings of the Seventh Heterogeneous Computing Workshop, p. 29. IEEE Computer Society (1998)
17. Cafaro, M., Aloisio, G. (eds.): Grids, Clouds and Virtualization. Springer (2010)
18. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008). DOI http://doi.acm.org/10.1145/1327452.1327492
19. Defanti, T., Foster, I., Papka, M.E., Kuhfuss, T., Stevens, R., Stevens, R.: Overview of the i-way: Wide area visual supercomputing. International Journal of Supercomputer Applications **10**(2), 123–130 (1996)

20. Dimitrakos Theoand Martrat, J., Wesner, S. (eds.): Service Oriented Infrastructures and Cloud Service Platforms for the Enterprise - A selection of common capabilities validated in real-life business trials by the BEinGRID consortium. Springer (2010)
21. Erwin, D.W., Snelling, D.F.: Unicore: A grid computing environment. In: Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing, pp. 825–834. Springer-Verlag, London, UK (2001)
22. Evangelinos, C., Hill, C.N.: Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's ec2. Cloud Computing and Its Applications (2008)
23. Foster, I.: What is the grid? a three point checklist (2002). URL `http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf`. "unpublished"
24. Foster, I.: Globus toolkit version 4: Software for service-oriented systems. In: IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp. 2–13 (2005)
25. Foster, I., Geisler, J., Nickless, B., Smith, W., Tuecke, S.: Software infrastructure for the i-way high-performance distributed computing experiment. In: In Proc. 5th IEEE Symp. on High Performance Distributed Computing, pp. 562–571. Society Press (1996)
26. Foster, I., Kesselman, C.: The globus project: a status report. Future Gener. Comput. Syst. **15**(5-6), 607–621 (1999). DOI http://dx.doi.org/10.1016/S0167-739X(99)00013-8
27. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid - enabling scalable virtual organizations. International Journal of Supercomputer Applications **15**, 2001 (2001)
28. Foster, I., Zhao, Y., Raicu, I., Lu, S.: Cloud computing and grid computing 360-degree compared. In: Grid Computing Environments Workshop, 2008. GCE '08, pp. 1 –10 (2008). DOI 10.1109/GCE.2008.4738445
29. Fox, G.C., Pierce, M.: Grids meet too much computing: Too much data and never too much simplicity (2007). URL `http://grids.ucs.indiana.edu/ptliupages/publications/TooMuchComputingandData.pdf`. "unpublished"
30. Gentzsch, W.: Sun grid engine: Towards creating a compute power grid. In: CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid, p. 35. IEEE Computer Society, Washington, DC, USA (2001)
31. Gottfrid, D.: URL `http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun`
32. Gray, J.: Distributed computing economics. Queue **6**(3), 63–68 (2008). DOI http://doi.acm.org/10.1145/1394127.1394131
33. Grimshaw, A.S., Wulf, W.A., The Legion Team, C.: The legion vision of a worldwide virtual computer. Commun. ACM **40**(1), 39–45 (1997). DOI http://doi.acm.org/10.1145/242857.242867
34. Hadoop: URL `http://hadoop.apache.org`
35. Hajdu, L., Kocoloski, A., Lauret, J., Miller, M.: Integrating Xgrid into the HENP distributed computing model. J. Phys. Conf. Ser. **119** (2008). DOI 10.1088/1742-6596/119/7/072018
36. Hughes, B.: Building computational grids with apple's xgrid middleware. In: ACSW Frontiers '06: Proceedings of the 2006 Australasian workshops on Grid computing and e-research, pp. 47–54. Australian Computer Society, Inc., Darlinghurst, Australia, Australia (2006)
37. Kesselman, C., Foster, I.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers (1998)
38. Lakshmi, J., Nandy, S.K.: Quality of Service for I/O Workloads in Multicore Virtualized Servers, chap. TBD, p. TBD. In: Cafaro and Aloisio [17] (2010)
39. Laszewski, G.V., Insley, J.A., Foster, I., Bresnahan, J., Kesselman, C., Su, M., Thiebaux, M., Rivers, M.L., Wang, S., Tieman, B., Mcnulty, I.: Real-time analysis, visualization, and steering of microtomography experiments at photon sources. In: Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing. SIAM, pp. 22–24 (1999)
40. Laure, E., Fisher, S.M., Frohner, A., Grandi, C., Kunszt, P.Z., Krenek, A., Mulmo, O., Pacini, F., Prelz, F., White, J., Barroso, M., Buncic, P., Hemmer, F., Di Meglio, A., Edlund, A.: Programming the grid using glite. Tech. Rep. EGEE-PUB-2006-029 (2006)

41. Laure, E., Jones, B.: Enabling grids for e-science: The egee project. Tech. Rep. EGEE-PUB-2009-001. 1 (2008)
42. Li, J., Loo, B.T., Hellerstein, J.M., Kaashoek, M.F., Karger, D.R., Morris, R.: On the feasibility of peer-to-peer web indexing and search. In: IPTPS, pp. 207–215 (2003)
43. Matsuoka, S., Shinjo, S., Aoyagi, M., Sekiguchi, S., Usami, H., Miura, K.: Japanese computational grid research project: Naregi. Proceedings of the IEEE **93**(3), 522–533 (2005)
44. Mersenne Research, I.: URL `http://www.mersenne.org`
45. Nanda, S., cker Chiueh, T.: A survey of virtualization technologies. Tech. rep., State University of New York at Stony Brook (2005)
46. Nanda, S., Li, W., Lam, L.C., Chiueh, T.c.: Bird: Binary interpretation using runtime disassembly. In: CGO '06: Proceedings of the International Symposium on Code Generation and Optimization, pp. 358–370. IEEE Computer Society, Washington, DC, USA (2006). DOI http://dx.doi.org/10.1109/CGO.2006.6
47. Perryman, A.L., Zhang, Q., Soutter, H.H., Rosenfeld, R., McRee, D.E., Olson, A.J., Elder, J.E., Stout, C.D.: Fragment-Based Screen against HIV Protease. Chemical Biology & Drug Design **75**(3), 257–268 (2010). DOI 10.1111/j.1747-0285.2009.00943.x
48. Schick, S.: URL `http://blogs.itworldcanada.com/shane/2008/04/22/five-ways-of-defining-cloud-computing/`
49. Skype: URL `http://www.skype.com`
50. Smarr, L., Catlett, C.E.: Metacomputing. Commun. ACM **35**(6), 44–52 (1992). DOI http://doi.acm.org/10.1145/129888.129890
51. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the condor experience. Concurrency - Practice and Experience **17**(2-4), 323–356 (2005)
52. Wang, L., Zhan, J., Shi, W., Liang, Y., Yuan, L.: In cloud, do mtc or htc service providers benefit from the economies of scale? In: MTAGS '09: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, pp. 1–10. ACM, New York, NY, USA (2009). DOI http://doi.acm.org/10.1145/1646468.1646475
53. Weiss, A.: Computing in the clouds. netWorker **11**(4), 16–25 (2007). DOI http://doi.acm.org/10.1145/1327512.1327513
54. Whitaker, A., Shaw, M., Gribble, S.D.: Denali: Lightweight virtual machines for distributed and networked applications. In: In Proceedings of the USENIX Annual Technical Conference (2002)
55. Xu, M.Q.: Effective metacomputing using lsf multicluster. Cluster Computing and the Grid, IEEE International Symposium on **0**, 100 (2001). DOI http://doi.ieeecomputersociety.org/10.1109/CCGRID.2001.923181

Quality of Service for I/O Workloads in Multicore Virtualized Servers

J. Lakshmi and S. K. Nandy

Abstract Emerging trend of multicore servers promises to be the panacea for all data-center issues with system virtualization as the enabling technology. System virtualization allows to create virtual replicas of the physical system, over which independent virtual machines can be created, complete with their own, individual operating systems, software and applications. This provides total system isolation of the virtual machines. Apart from this, the key driver for virtualization adoption in data-centers will be safe virtual machine performance isolation that can be achieved over a consolidated server with shared resources. This chapter identifies the basic requirements for performance isolation of virtual machines on such servers. The consolidation focus is on enterprise workloads that are a mix of compute and I/O intensive workloads. An analysis of prevalent, popular system virtualization technologies is presented with a view towards application performance isolation. Based on the observed lacunae, an end-to-end system virtualization architecture is proposed and evaluated.

1 Introduction

System virtualization on the emerging multicore servers is a promising technology that has solutions for many of the key data-center issues. Today's data-centers have concerns of curtailing space and power footprint of the computing infrastructure, which the multicore servers favorably address. A typical multicore server has sufficient computing capacity for aggregating several of server applications on a single

J. Lakshmi
SERC, Indian Institute of Science, Bangalore-560012, India, e-mail: jlakshmi@serc.iisc.ernet.in

Prof. S. K. Nandy
SERC, Indian Institute of Science, Bangalore-560012, India e-mail: nandy@serc.iisc.ernet.in

physical machine. The most significant issue with co-hosting multiple server applications on a single machine is with the software environment of each of the applications. System virtualization addresses this problem since it enables the creation of virtual replicas of a complete system, over which independent virtual machines (VMs) can be built, complete with their own, individual operating systems, software and applications. This results in complete software isolation of the VMs, which allows independent applications to be hosted within independent virtual machines on a single physical server.

Apart from the software isolation, the key driver for virtualization adoption in data-centers will be safe virtual machine performance isolation that can be achieved over a consolidated server. This is essential, particularly for enterprise application workloads, like database, mail, and web-based applications, which have both CPU and I/O workload components. Current commodity multicore technologies have system virtualization architectures that provide CPU workload isolation. The number of CPU-cores in comparison to I/O interfaces is high in multicore servers. This results in the sharing of I/O devices among independent virtual machines. As a result, this changes the I/O device sharing dynamics when in comparison to dedicated servers, wherein all the resources like the processors, memory, I/O interfaces for disk and network access are architected to be managed by a single OS. On such systems, solutions that optimize or maximize the application usage of the system resources are sufficient to address the performance of the application. When multiple, independent applications are consolidated on to a multicore server, using virtual machines, performance interference caused due to shared resources across multiple VMs adds to the performance challenges. The challenge is in ensuring performance of the independent I/O intensive applications, hosted inside isolated VMs, on the consolidated server while sharing a single I/O device [11].

Prevalent virtualization architectures suffer from the following distinct problems with regard to I/O device virtualization;

1. Device virtualization overheads are high due to which there is a reduction in the total usable bandwidth by an application hosted inside the VM.
2. Prevalent device virtualization architectures are such that sharing of the device causes its access path also to be shared. This causes performance degradation that is dependent on I/O workloads and limits scalability of VMs that can share the I/O device [1].
3. Device access path sharing causes security vulnerabilities for all the VMs sharing the device [36].

These reasons cause variability in application performance that is dependent on the nature of consolidated workloads and the number of VMs sharing the I/O device.

One way to control this variability is to impose necessary Quality of Service (QoS) controls on resource allocation and usage of shared resources. Ideally, the QoS controls should ensure that:

- There is no loss of application performance when hosted on virtualized servers with shared resources.
- Any spare resource is made available to other contending workloads.

The chapter starts with a discussion on the resource specific QoS controls that an application's performance depends on. It then explores the QoS controls for resource allocation and usage in prevalent system virtualization architectures. The focus of this exploration is on the issues of sharing a single NIC across multiple virtual machines (VMs). Based on the observed lacunae, an end-to-end architecture for virtualizing network I/O devices is presented. The proposed architecture is an extension to that of what is recommended in the PCI-SIG IOV specification [22]. The goal of this architecture is to enable fine-grained controls to a VM on the I/O path of a shared device leading to minimization of the loss of usable device bandwidth without loss of application performance. The proposed architecture is designed to allow native access of I/O devices to VMs and provides the device level QoS controls for managing VM specific device usage. The architecture evaluation is carried out through simulation on a layered queuing network(LQN) [3, 4] model to demonstrate its benefits. The proposed architecture improves application throughput by 60% as in comparison to what is observed on the existing architectures. This performance is closer to the performance observed on non-virtualized servers. The proposed I/O virtualization architecture meets its design goals and also improves the number of VMs that can share the I/O device. Also, the proposed architecture eliminates some of the shared device associated security vulnerabilities [36].

2 Application requirements for performance isolation on shared resources

Application performance is based on timely availability of the required resources like processors, memory and I/O devices. The basic guideline for consolidating enterprise servers over multicore virtualized systems is by ensuring availability of required resources as and when required [7]. For the system to be able to do so, the application resource requirements are enumerated using resource requirement (RR) tuples. A RR tuple is an aggregated list of various resources that the application's performance depends on. Thus RR tuple is built using individual resource tuples. Each resource tuple is made up of a list of resource attributes or the attribute tuples. Using this definition, a generic RR tuple can be written as follows:

$$
\begin{aligned}
&Application(RR) = \\
&(R1{<}A1(Unit, Def, Min, Max), A2(Unit, Def, Min, Max), ...{>}, \\
&R2{<}A1(Unit, Def, Min, Max), A2(Unit, Def, Min, Max), ...{>}, \\
&...)
\end{aligned}
$$

where:

- Application(RR) - Resource requirement tuple of the application.
- R1 - Name of a resource, viz. processor (CPU), memory, network(NIC), etc.

- A1 - Name of the attribute of the associated resource. As an example, if A1 represents the CPU speed attribute, it is denoted by the tuple that describes the CPU speed requirements for the application.
- (Unit, Def, Min, Max) - represent the *Unit* of measurement, *Default*, *Minimum* and *Maximum* values of the resource attribute.

Using the XML format for resource specification, akin to Globus Resource Specification Language [14], the following example in Figure 1 illustrates the application(RR) for a typical VM that has both compute and I/O workloads.

```
<Application_RR_descriptor>
<CPU_Resource_Descriptor>
<Speed>
        <Unit>MHz</Unit>
        <Default>1800</Default>
        <Minimum>1500</Minimum>
        <Maximum>2000</Maximum>
</Speed>
<NCPU>
        <Default>4</Default>
        <Minimum>1</Minimum>
        <Maximum>4</Maximum>
</NCPU>
<L1Cache>
        <Unit>KB</Unit>
        <Default>64</Default>
        <Minimum>64</Minimum>
        <Maximum>64</Maximum>
</L1Cache>
</CPU_Resource_Descriptor>
<Memory_Resource_Descriptor>
<Size>
        <Unit>MB</Unit>
        <Default>2000</Default>
        <Minimum>1000</Minimum>
        <Maximum>4000</Maximum>
</Size>
<Bandwidth>
        <Unit>MBps</Unit>
        <Default>6400</Default>
        <Minimum>6400</Minimum>
        <Maximum>6400</Maximum>
</Bandwidth>
</Memory_Resource_Descriptor>
<Network_Resource_Descriptor>
<Speed>
        <Unit>Mbps</Unit>
        <Default>1000</Default>
        <Minimum>100</Minimum>
        <Maximum>1000</Maximum>
</Speed>
<Bandwidth>
        <Unit>KBps</Unit>
        <Default>5000</Default>
        <Minimum>5000</Minimum>
        <Maximum>8000</Maximum>
</Bandwidth>
        <Unit>KB</Unit>
        <Default>64</Default>
        <Minimum>64</Minimum>
        <Maximum>64</Maximum>
</Bandwidth>
</Network_Resource_Descriptor>
<Disk_Resource_Descriptor>
<Size>
        <Unit>MB</Unit>
        <Default>1000</Default>
        <Minimum>1000</Minimum>
        <Maximum>1000</Maximum>
</Size>
<Bandwidth>
        <Unit>MBps</Unit>
        <Default>100</Default>
        <Minimum>100</Minimum>
        <Maximum>400</Maximum>
</Bandwidth>
</Disk_Resource_Descriptor>
</Application_RR_Descriptor>
```

Fig. 1 An example Application Resource Requirement tuple for a VM, expressed in XML.

In the depicted example, the resource tuple for the CPU resource is described by the $<$*CPU_Resource_Descriptor*$>$ and $<$*/CPU_Resource_Descriptor*$>$ tag pair. Attribute tuples relevant to and associated with this resource are specified using the attribute and value tag pair, within the context of resource tag pair. Each attribute is specified by its *Unit of measurement*, *Default*, *Minimum* and *Maximum* values that the virtual machine monitor's (VMM's) resource allocator uses for allocating the resource to the VM. In the example, the CPU speed is defined by the attribute tags $<$*Speed*$>$ and $<$*/Speed*$>$. The *Unit of Measurement* for CPU speed is mentioned as MHz. The attribute values for *Default*, *Minimum* and *Maximum* specify the CPU speed required for the desired application performance hosted inside the VM. *Default* value specifies the attribute value that the VMM can initially allocate to the VM. On an average, this is the value that the VM is expected to use. The *Minimum* value defines the least value for the attribute that the VM needs to support the guaranteed application performance. And *Maximum* value defines the

maximum attribute value that the VM can use while supporting its workload. All the three attribute values can be effectively used if the VMM uses dynamic adaptive resource allocation policies. For each resource, its tuple is specified using attribute value tuples that completely describe the specific resource requirement in terms of the quantity, number of units, and speed of resource access.

On a virtualized server the physical resources of the system are under the control of the VMM. The resource tuples are used by the VMM while allocating or deallocating resources to the VMs. It can be assumed that RR contains values that are derived from the application's performance requirements. In the context of multicore servers, with server consolidation as the goal, each application can be assumed to be hosted in an independent VM which encapsulates the application's environment. Hence, the application's resource tuples can be assumed to be the RR for each VM of the virtualized server. In the case where multiple applications are co-hosted on a single VM, these resource tuples can be arrived at by aggregating the resource requirements of all the applications hosted by the VM.

3 Prevalent Commodity Virtualization technologies and QoS controls for I/O device sharing

Commodity virtualization technologies like *Xen* and *Vmware* have made the normal desktop very versatile. A generic architecture of system virtualization, implemented in these systems, is given in Figure 1. The access to CPU resource is native, to all

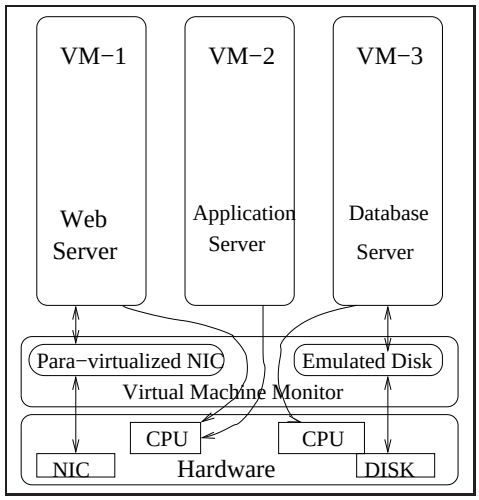


Fig. 2 Generic System Virtualization architecture of prevalent commodity virtualization technologies.

VMs sharing the CPU, for all instructions except the privileged instructions. The privileged instructions are virtualized, i.e., whenever such instructions are executed

from within the VM, they are trapped and control is passed to the VMM. All I/O instructions fall under the category of privileged instructions. Thus, I/O devices like the Network Interface Card (NIC) and the DISK are treated differently, when virtualized. There are two different, popularly adopted, methods used for virtualizing I/O devices, namely, para-virtualization and emulation [28]. Para-virtualized mode of access is achieved using a virtual device driver along with the physical device driver. A hosting VM or the VMM itself has exclusive, native access to the physical device. Other VMs sharing the device use software based mechanisms, like the virtual device driver, to access the physical device via the hosting VM or the VMM. In emulated mode of access, each VM sharing the physical device has a device driver that is implemented using emulation over the native device driver hosted by the VMM. Both these modes provide data protection and integrity to independent VMs but suffer from loss of performance and usable device bandwidth. Details of the evaluation are elucidated in the following section. In order to understand the effect of the device virtualization architectures on application performance, experimental results of well-known benchmarks, *httperf* [8] and *netperf* [2], are evaluated. The first experiment is described in section 3.1 and explores how virtualization affects application performance. The second experiment, described in section 3.2, evaluates the existing QoS constructs in virtualized architectures for their effectiveness in providing application specific guarantees.

3.1 Effect of Virtualization on Application Performance

Prevalent commodity virtualization technologies, like *Xen* and *Vmware*, are built over system architectures designed for single OS access. The I/O device architectures of such systems do not support concurrent access to multiple VMs. As a result, the prevailing virtualization architectures support I/O device sharing across multiple VMs using software mechanisms. The result is device sharing along with its access path. Hence, serialization occurs at the device and within the software layers used to access the device.

In virtualized servers, disk devices are shared differently compared to sharing of NICs. In the case of disk devices, a disk partition is exposed as a file-system that is exported to a single VM. Any and every operation to this file-system is from a single VM and all read and write disk operations are block operations. The data movements to and from the filesystem is synchronized using the file-system buffer cache that is resident within the VM's address space. The physical data movement is co-ordinated by the native device drivers within the VMM or the hosted VM, and the para-virtualized or emulated device driver resident in the VM. In the para-virtualized mode, the overheads are due to the movement of data between the device hosting VM and the application VM. And in the case of emulation mode of access, the overheads manisfest due to the translation of every I/O instruction between the emulated device driver and the native device driver. Due to this nature of I/O activity, VM specific file-system policies get to be implemented within the software layers

of the VMM or the hosting VM. Since the file-system activity is block based, setting up appropriate block sizes can, to some extent, enable the control of bandwidth and speed requirements on the I/O channel to the disk. However, these controls are still coarse-grained and are insufficient for servers with high consolidation ratios.

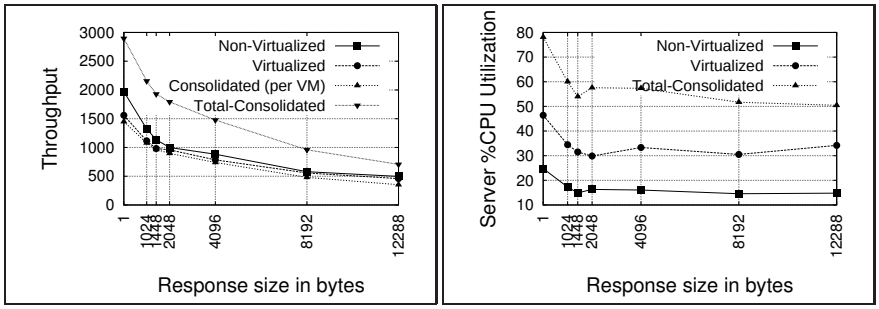
For network devices the existing architecture poses different constraints. Unlike for the disk I/O which is block based, network I/O is packet based and sharing a single NIC with multiple VMs has intermixed packet streams. This intermixing is transparent to the device and is sorted into per VM stream by the VMM or the hosting VM. Apart from this, every packet is subjected to either instruction translation (emulation) or address translation (para-virtualization) due to virtualization. In both the cases, virtualization techniques build over existing "single-OS over single hardware" model. This degrades application performance.

Throughput studies of standard enterprise benchmarks highlight the effects of virtualization and consolidation based device sharing. Since NIC virtualization puts forth the basic issues with virtualization technologies, an analysis of NIC sharing over application throughput is presented. Figures 3a and 4a depict the performance of two standard benchmarks, *netperf* [2] and *httperf* [8] wherein the benchmark server is hosted in three different environments, namely non-virtualized, virtualized and consolidated server. The non-virtualized environment is used to generate the baseline for the metric against which the comparison is made for the performance on virtualized and consolidated server. The virtualized server hosts only one VM wherein the complete environment of the non-virtualized server is reproduced inside the VM. This environment is used, to understand the overheads of virtualization technology used. The consolidated server hosts two VMs, similar to the VM of the virtualized server, but with both VMs sharing the same NIC. The consolidated server environment is used to understand the I/O device sharing dynamics on a virtualized server.

For the *netperf* benchmark, *netperf* is the name of the client and *netserver* is the server component. The study involves execution of the TCP_CRR test of *netperf*. The TCP_CRR test measures the connect-request-response sequence throughput achievable on a server and is similar to the access request used in *http* based applications. In the case of *httperf* benchmark, the client, called *httperf*, communicates with a standard *http* server using the *http* protocol. In the *httperf* test used, the client allows for specifying the workload in terms of the number of *http* requests to the server in one second, for a given period of time, to generate statistics like average number of replies received from the server (application throughput), average response time of a reply and the network bandwidth utilized for the workload. While *netperf* gives the achievable or achieved throughput, *httperf* gives an average throughput calculated for a subset of samples, executed over a specified period of time, within the given experiment. Hence, *httperf* results give an optimistic estimate which may fall short of expectation in situations where sustained throughput is a requirement.

It is observed from the throughput graphs of *netperf* and *httperf*, that there is a significant drop in application throughput as it is moved from non-virtualized to *Xen* virtualized server. *Xen* virtualization uses para-virtualization mechanism with

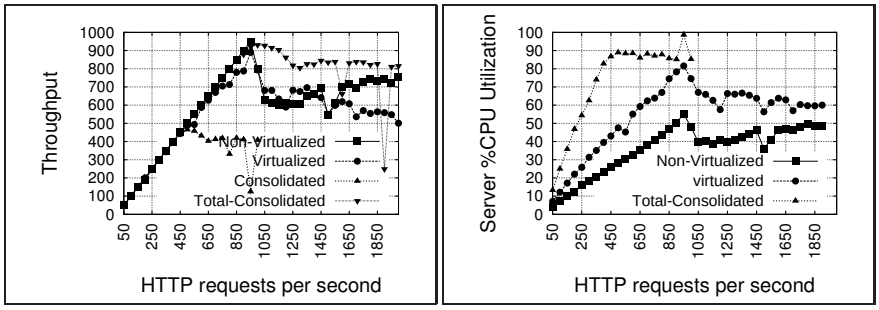
software bridging to virtualize the NIC. The application throughput loss is the overall effect of virtualization overheads. There is a further drop when the application is hosted on a consolidated server with the VMs sharing the NIC. This is obvious, since for the consolidated server, the NIC is now handling twice the amount of traffic in comparison to that of the virtualized server case. It is interesting to note that the virtualization overheads manifest as extra CPU utilization on the virtualized server [18]. This is observed by the CPU utilization graphs of Figures 3b and 4b. Both the benchmarks indicate increased CPU activity to support the same application throughput. This imposes response latencies leading to application throughput loss and also usable device bandwidth loss for the VM. The noteworthy side effect of this device bandwidth loss, for a VM, is that, it is usable by another VM, which is sharing the device. This is noticed in the throughput graphs of the consolidated server for *netperf* benchmark. It is an encouraging fact for consolidating I/O workloads on virtualized servers. However, *httperf* benchmark performance on the consolidated server is not very impressive and suggests further investigation.



(a) Server achievable Throughput. (b) Server CPU Utilization.

Fig. 3: *netserver* achievable Throughput and corresponding %CPU Utilization on *Xen* virtualized platform.

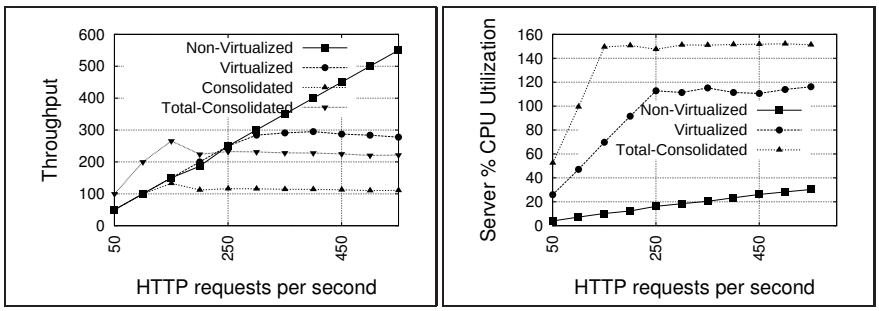
Conducting this experiment on *Vmware* virtualization technology produces similar behavior, which is depicted in Figure 5. The *httperf* benchmark tests are conducted on an Intel Core2Duo server with two cores. Unlike the case of *Xen*, pinning of ESXi server (the hypervisor) to a CPU is not allowed. Hence, any CPU utilization measurements for the ESXi hypervisor on *Vmware* show utilizations for all CPUs included. This results in %CPU utilization above 100% in the case of multicore systems. *Vmware- ESXi* server implements NIC virtualization using device emulation. It is observed that the overheads of emulation are comparatively quite high in relation to para-virtualization used in *Xen*. Here also, virtualization of NIC results in using up more CPU to support network traffic on a VM when in comparison to a non-virtualized server. The other important observation is the loss of application throughput. Device emulation imposes higher service times for packet processing and hence drastic drop of application throughput is observed, when in compari-



(a) Server Throughput (b) Server CPU Utilization

Fig. 4: *httperf* server Throughput and % CPU Utilization on *Xen* virtualized single-core server. The hypervisor and the VMs are pinned to the same core.

son to non-virtualized and para-virtualized systems. In this case 70% drop on the maximum sustained throughput is observed when in comparison to the throughput achieved in the non-virtualized environment. This loss is visible even in the consolidated server case. Interestingly, the total network bandwidth used in the case of consolidated VMs on *Vmware-ESXi* was only 50% of the available bandwidth. Hence, the bottleneck is the CPU resource available to the VMs, since each of the VM was hosted on the same core. It is reasonable to believe that multicores can al-

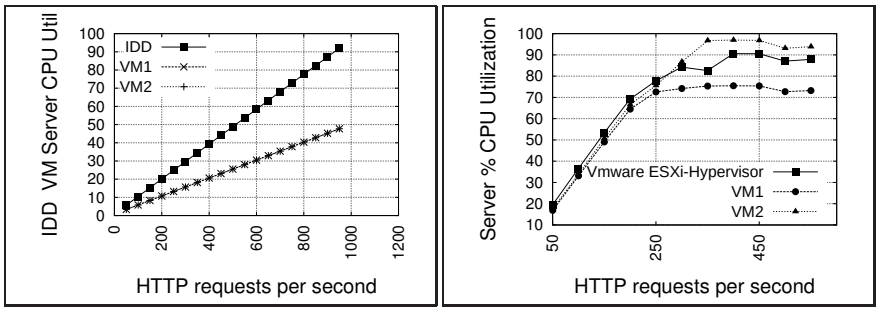


(a) Server Throughput (b) Server CPU Utilization

Fig. 5: *httperf* server Throughput and % CPU Utilization on *Vmware-ESXi* virtualized single-core server. The VMs are pinned to a single core while hypervisor uses all available cores.

leviate the CPU requirement on the consolidated server. On such systems, the CPU requirement of the VMs can be decoupled from that of the VMM by allocating different CPU cores to each of them. Study of *httperf* benchmark on consolidated

server with each VM pinned to a different core, for both *Xen* and *Vmware-ESXi* virtualized server show otherwise. Application throughput increase is observed when in comparison to single core consolidated server, but this increase still falls short



(a) Xen Server % CPU Utilization (b) Vmware-ESXi Server % CPU Utilization

Fig. 6: *httperf* server % CPU Utilization on *Xen* and *Vmware-ESXi* virtualized multicore server. The hypervisor and the VMs are pinned to independent cores.

by 10% of what was achieved for the non-virtualized server. The reason for this shortcoming is because both VMs sharing the NIC also share the access path that is implemented by the Independent Driver Domain(IDD) in the case of *Xen* and the hypervisor in the case of *Vmware-ESXi* virtualized server. This sharing manifests as serialization and increased CPU utilization of the IDD or the hypervisor, which becomes the bottleneck as the workload increases. Also, this bottleneck restricts the number of VMs that can share the NIC. This is clearly depicted in the graphs of Figure 6. In Figure 6a, it is observed that as the *httperf* workload is increasing, there is a linear increase in the CPU utilization of the VMs as well as the Xen-IDD hosting the NIC. The CPU utilization of the IDD, however, is much more when compared to the CPU utilization of either of the VMs. This is because the IDD is supporting network streams to both the VMs. As a consequence, it is observed that even though there is spare CPU available to the VMs, they cannot support higher throughput since the IDD has exhausted its CPU resource. This indicates that lack of concurrent device with concurrent access, imposes serialization constraints on the device and its access path which limits device sharing scalability on virtualized servers. This behavior is also observed in the case of the *Vmware-ESXi* server as is depicted in Figure 6b. However, as in the case of single core experiments, the CPU Utilization by the hypervisor and the VMs is significantly much higher when in comparison to the *Xen* server for the same benchmark workload. This results in poor performance when compared to para-virtualized devices, but yeilds more unused device bandwidth. As a result *Vmware-ESXi* server supports higher scalability for sharing the NIC.

The analysis for multicore virtualized server CPU Utilization indicates that even with the availability of required resources, for each of the VMs and the hypervisor, the device sharing architecture has constraints that impose severe restrictions

in usable bandwidth and scalability of device sharing. These constraints are specifically due to serialization of device and its access paths. Hence, it is necessary to re-architect device virtualization to enable concurrent device access to eliminate the bottlenecks evident in device sharing by the VMs.

3.2 Evaluation of Network QoS Controls

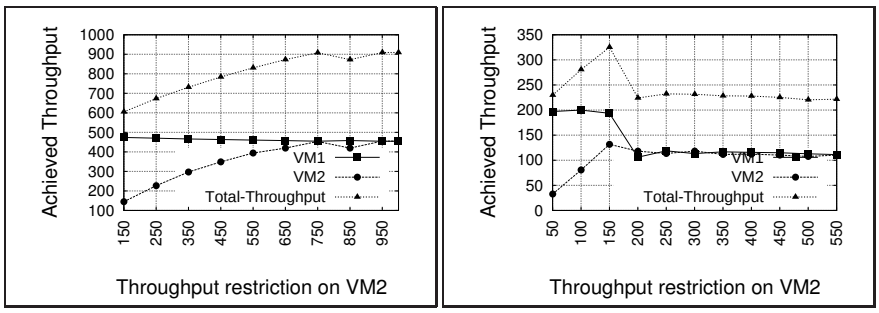
The most noteworthy point of observation of the study in section 3.1 is the behavior of each stream of benchmark on the consolidated server. In general, it is observed that there is a further reduction of throughput on the consolidated server, when in comparison to the single VM on a virtualized server, for both the benchmarks, *netperf* and *httperf*, with a marked decrement in the latter case. This indicates the obvious; lack of QoS constraints would lead to severe interference in performance delivered by the device sharing VMs.

The current commodity virtualization technologies like *Xen* and *Vmware* allow for VM specific QoS controls on different resources using different mechanisms. The CPU resource allocations are handled directly by the VMM schedulers like Credit, SEDF or BVT schedulers of *Xen* [12]. Also, as discussed in [18, 25, 33, 32] the existing CPU resource controls are fine-grained enough to deliver desired performance for CPU-based workloads. The problem is with I/O devices. The access to an I/O device is always through the hypervisor or the driver domain OS kernel to ensure data integrity and protection. The device is never aware as to which VM is using it at any given instance of time; this information and control is managed by the hypervisor or the driver domain. Hence, resource allocation controls with regard to the I/O devices are at a higher abstraction level rather than at the device level, unlike in the case of the CPU resource. These controls are effective for the outgoing streams from the server, since packets that overflow are dropped before reaching the NIC. But for the incoming stream, the control is ineffective since the decision of accepting or rejecting is made after the packet is received by the NIC. Hence, the controls are coarse-grained and affect the way resource usage is controlled and thereby the application performance. In scenarios where I/O device utilization is pushed to its maximum, limitations of such QoS controls are revealed as loss of usable bandwidth or scalability of sharing thereby causing unpredictable application performance, as is illustrated in the next section.

To understand the effect of software based QoS controls for network bandwidth sharing, an experimental analysis of *httperf* benchmark on a consolidated server is presented. The consolidated server hosts two VMs, namely VM1 and VM2, that are sharing a NIC. Each VM hosts one *http* server that responds to a *httperf* client. The *httperf* benchmark is chosen for this study because it allows customization of observation time of the experiment. This is necessary since the bandwidth control mechanisms that are available are based on time-sampled averages and hence, need a certain interval of time to affect application throughput. The experiment involves two studies, one is that of best effort sharing where no QoS is imposed on either

of the VMs and in the second case VM1 is allowed to use the available network bandwidth when VM2 is constrained, by imposing specific QoS value based on the desired application throughput. For both studies, each VM is subjected to equal load from the *httperf* clients.

The performance of consolidated server corressponding to the best effort sharing case is presented in Figure 4a and Figure 5a. As it is observed from the graphs, the NIC bandwidth sharing is equal in both the virtualization solutions. When no QoS controls are enforced and each VM has equal demand for the resource, it is shared equally on a best effort basis. In the second study, when bandwidth control is enforced on the VM2, while allowing complete available bandwidth to the VM1, the expected behavior is to see improved throughput for the unconstrained VM1. This is to say, that VM1 performance is expected to be better when in comparison to the best effort case. Figure 7 demonstrates that imposing QoS controls on VM2 does not translate to extra bandwidth availability for the other, unconstrained VM. The reasons for this behavior are a multitude. The most significant ones being the virtualization overhead in terms of the CPU resource required by the VMM or the hosting VM to support I/O workload, serialization of the resource and its access path, lack of control on the device for the VM specific incoming network stream and lastly, higher priority to the incoming stream over the outgoing stream at the device. All these lead to unpredictable application performance in-spite of applying appropriate QoS controls. Also, it is interesting to note that the variation in performance is dependent on the nature of the consolidated workloads. This performance variation affects all the consolidated workloads and makes the application guarantee weak. On multicore servers hosting many consolidated workloads of a datacenter, indeterminate performance is definitely not acceptable. Also, since virtual device is an abstraction supported in software, device usage controls are coarse grained and hence ineffective. This could lead to an easy denial of service attack on a consolidated server with shared devices.



(a) *Xen-IDD Linux* based QoS controls on VM2 (b) *Vmware-ESXi* QoS controls on VM2

Fig. 7: Effect of hypervisor network bandwidth controls on application throughput for consolidated virtualized server hosting two VMs.

The bandwidth controls enforced are based on the following principle. For each of the virtualization technologies used, i.e. *Xen* and *Vmware*, the network bandwidth used by a single VM to support different *httperf* request rates, without performance loss, is measured. These bandwidth measurements are used to apply control on the outgoing traffic from VM2. Currently, the available controls allow constraints only on the outgoing traffic. On the incoming traffic, ideally the control should be applied at the device so that any packet causing overflow is dropped before reception. Such controls are not available at present. Instead, in *Xen*, at-least one can use the *netfilter module's* stream based controls after receiving the packet. This does not serve the purpose, because by receiving a packet that could potentially be dropped later, the device bandwidth is anyway wasted. Hence, the study involves using only the outgoing traffic controls for the constrained VM.

The selection of different range of workloads, for each of the virtualized server, is based on the maximum throughput that each can support in a consolidated server environment. For each QoS control, the maximum throughput achieved, without loss, by each of the VM, is plotted in the graphs of Figures 7a and 7b. In these Figures, the x-axis represents the *httperf* request rate based on which the network bandwidth control was applied on the VM2 and the y-axis represents the application throughput achieved by each of the VMs. In the case of *Xen*, *Linux tc* utility of the *netfilter* module [35] is used to establish appropriate bandwidth controls. Specifically, each traffic stream from the VMs is defined using *htb* class with *tbf* queue discipline with the desired bandwidth control. Each queue is configured with a burst value to support a maximum of 10 extra packets. In the case of *Vmware-ESXi* server, the *Veam Monitor* controls for network bandwidth are used and populated with the same QoS controls as is done for the *Xen* server.

Based on the behavior of the benchmarks, following bottlenecks are identified for sharing network I/O device across multiple VMs on *Xen* or *Vmware-ESXi* virtualized server.

- Virtualization increases the device utilization overheads, which leads to increased CPU utilization of the hypervisor or the IDD hosting the device.
- Virtualization overheads cause loss of device bandwidth utilization from inside a VM. Consolidation improves the overall device bandwidth utilization but further adds to CPU utilization of the VMM and IDD. Also, if the VMM and IDD do not support concurrent device access APIs they themselves become the bottlenecks for sharing the device.
- QoS features for regulating incoming and outgoing traffic are currently implemented in the software stack. Uncontrolled incoming traffic at the device, to a VM, that is sharing a network device, can severely impact the performance of other VMs because the decision to drop an incoming packet is taken after the device has received the packet. This could potentially cause a denial of service attack on the VMs sharing the device.

Based on the above study, a device virtualization architecture is proposed and described in the following sections. The proposal is an extension to I/O virtualization architecture, beyond what is recommended by the PCI-SIG IOV specification [22]. The PCI-SIG IOV specification defines the rudiments for making I/O devices virtualization aware. On the multicore servers with server consolidation as the goal, particularly in the enterprise segment, being able to support multiple virtual I/O devices on a single physical device is a necessity. High speed network devices, like 10Gbps NICs, are available in the market. Pushing such devices to even 80% utilization needs fine-grained resource management at the device level. The basic goal of the proposed architecture is to be able to support finer levels of QoS controls, without compromising on the device utilization. The architecture is designed to enable native access of I/O devices to the VMs and provide device level QoS hooks for controlling VM specific device usage. The architecture aims to reduce network I/O device access latency and enable improvement in effective usable bandwidth in virtualized systems by addressing the following issues:

- Separating device management issues from device access issues.
- Allowing native access of a device to a VM by supporting concurrent device access and eliminating hypervisor/IDD from the path of device access.
- Enable fine-grained resource usage controls at the device.

In the remaining part of the chapter, we bring out the need for extending I/O device virtualization architecture in section 4. Section 5 highlights the issues in sharing of the I/O device and its access path in prevalent virtualization architectures leading to a detailed description of the proposed architecture to overcome the bottlenecks. *Xen* virtualization architecture is taken as the reference model for the proposed architecture. In the subsequent part of the section, complete description of the network packet work-flow for the proposed architecture is presented. These work-flows form the basis for generating the LQN model, that is used in the simulation studies for architecture evaluation described in section 6. Brief description of the LQN model generation and detailed presentation of simulation results is covered in section 7. Finally, in section 8 the chapter conclusion highlights on the benefits of the architecture.

4 Review of I/O virtualization techniques

Virtualization technologies encompass a variety of mechanisms to decouple the system architecture and the user-perceived behavior of hardware and software resources. Among the prevalent technologies, there are two basic modes of virtualization, namely, full system virtualization as in *Vmware* [16] and para-virtualization as in *Xen* [12]. In full system virtualization complete hardware is replicated virtu-

ally. Instruction emulation is used to support multiple architectures. The advantage of full system virtualization is that it enables unmodified Guest operating systems (GuestOS) to execute on the VM. Since it adopts instruction emulation, it tends to have high performance overheads as observed in the experimental studies described earlier. In Para-virtualization the GuestOS is also modified suitably to run concurrently with other VMs on the same hardware. Hence, it is more efficient and offers lower performance overheads. In either case, system virtualization is enabled by a layer called the virtual machine monitor (VMM), also known as the hypervisor, that provides the resource management functionality across multiple VMs. I/O virtualization started with dedicated I/O devices assigned to a VM and has now evolved to device sharing across multiple VMs through virtualized software interfaces [28]. A dedicated software entity, called the I/O domain is used to perform physical device management. The I/O domain can be part of the VMM or be an independent domain, like the independent driver domain (IDD) of *Xen*. In the case of IDD the I/O devices are private to the domain and memory accesses by the devices are restricted to the IDD. Any application in a VM seeking access to the device has to route the request through the IDD and the request has to pass through the address translation barriers of the IDD and VM [20, 23].

Recent publications on concurrent direct network access (CDNA) [24] and scalable self-virtualizing network interface [17] are similar to the proposed work in the sense that they explore the I/O virtualization issues on the multicore platforms and provision for concurrent device access. However, the scalable self-virtualizing interface describes assigning a specific core for network I/O processing on the virtual interface and exploits multiple cores on embedded network processors for this. The authors do not detail how the address translation issues are handled, particularly in the case of virtualized environments. CDNA is architecturally closer to our architecture since it addresses concurrent device access by multiple VMs. CDNA relies on per VM Receive (Rx) and Transmit (Tx) ring buffers to manage VM specific network data. The VMM handles the virtual interrupts and the *Xen* implementation still uses IDD to share the I/O device. Also, authors do not address the performance interference due to uncontrolled data reception by the device nor do they discuss the need for addressing the QoS controls at the device level.

The proposed architecture addresses these and suggests pushing the basic constructs to assign QoS attributes like required bandwidth and priority into the device to get fine-grained control on interference effects. Also, the proposed architecture has it basis in *exokernel's* [6] philosophy of separating device management from protection. In *exokernel*, the idea was to extend native device access to applications with the *exokernel* providing the protection. In the proposed approach, the extension of native device access is with the VM, the protection being managed by the VMM and the device collectively. A VM is assumed to be running the traditional GuestOS without any modifications with native device drivers. This is a strong point in support of legacy environments without any need for code modification. Further, the PCI-SIG community has realized the need for I/O device virtualization and has come out with the IOV specification to deal with it. The IOV specification, however, talks about device features to allow native access to virtual device interfaces,

through the use of I/O page tables, virtual device identifiers and virtual device specific interrupts. The specification presumes that QoS is a software feature and does not address this. Many implementations adhering to the IOV specification are now being introduced in the market by Intel [19], Neterion [26], NetXen [27], Solarflare [34], etc. Apart from these, the Crossbow [29] suite from SUN Microsystems talks about this kind of resource provisioning. However, Crossbow is a software stack over a standard IOV complaint hardware. The results published using any of these products are exciting in terms of the performance achieved. These devices when used within the prevalent virtualization technologies need to still address the issue of provisioning QoS controls on the device. Lack of such controls, as illustrated by the previously described experimental studies, cause performance degradation and interference that is dependent on the workloads sharing the device.

5 Enhancement to I/O virtualization architecture

The analysis of prevalent commodity virtualization technologies in section 3 clearly highlights the issues that need to be addressed while sharing I/O devices across independent VMs on multicore virtualized servers. It is also observed that while para-virtualization offers better performance for the application, emulation is the alternative for improved consolidation. The goals are seemingly orthogonal since current technologies build over virtualization unaware I/O devices. The proposed architecture takes a consolidated perspective of merging these two goals, that of ensuring application performance without losing out on the device utilization by taking advantage of virtualization aware I/O devices and re-architecting the end-to-end virtualization architecture to deliver the benefits. In order to understand the benefits of the proposed architecture the *Xen* based para-virtualization architecture for I/O devices is taken as the reference model. In the existing *Xen* virtualization architecture, analysis of the network packet work-flow highlights following bottlenecks:

- Since the NIC device is shared, the device memory behaves like a common memory for all the contending VMs accessing the device. One misbehaving VM can ensure deprivation leading to data loss for another VM.
- The *Xen-IDD* is the critical section for all the VMs sharing the device. IDD incurs processing overheads for every network operation executed on behalf of each VM. Current IDD implementations do not have any hooks for controlling the overheads on per VM basis. Lack of such controls lead to performance interference in the device sharing VMs.
- Every network packet has to cross the address translation barrier of VMM to IDD to VM and vice-versa. This happens because of lack of separation of device management issues from device access issues. Service overheads of this stage-wise data movement cause drop in effective utilized device bandwidth. In multicore servers with scarce I/O devices, this would mean having high-bandwidth underutilized devices and low throughput applications on the consolidated server.

To overcome the above listed drawbacks, the proposed architecture enhances I/O device virtualization to enable separation of device management from device access. This is done by building device protection mechanisms into the physical device and managed by the VMM. As an example, for the case of NIC, the VMM recognizes the destination VM of an incoming packet by the interrupt raised by the device and forwards it to the appropriate VM. The VM then processes the packet as it would do so in the case of non-virtualized environment. Thus, device access and scheduling of device communication are managed by the VM that is using it. The identity for access is managed by the VMM. This eliminates the intermediary VMM/IDD on the device access path and reduces I/O service time, which improves the application performance on virtualized servers and also the usable device bandwidth which results in improved consolidation. In the following subsections we describe the NIC I/O virtualization architecture, keeping the above goals in mind, and suggest how the system software layers of the VMM and the GuestOS inside the VM should use the NIC hardware that is enabled for QoS based concurrent device access.

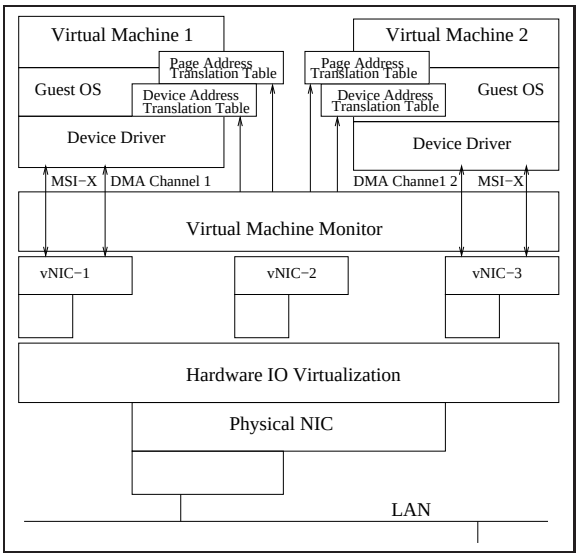


Fig. 8 NIC architecture supporting independent reconfigurable virtual-NICs.

5.1 Proposed I/O Virtualization Architecture Description

Figure 8 gives a block schematic of the proposed I/O virtualization architecture. The picture depicts a NIC card that can be housed within a multicore server. The card has a controller that manages the DMA transfer to and from the device memory. The standard device memory is now replaced by a partitionable memory supported with *n* sets of device registers. A set of *m* memory partitions, where

$$m \leq n$$

along-with device registers form the virtual-NICs (vNICs). The device memory is reconfigurable, i.e. dynamically partitionable, and the VM's QoS requirements drive the sizing of the memory partition of a vNIC. The advantage of having a dynamically partitionable device memory is that any unused memory can be easily extended into or reduced from a vNIC in order to support adaptive QoS specifications. The NIC identifies the destination VM of an arriving packet, based on the logical device address associated with it. A simple implementation is to allow a single physical NIC to support multiple MAC address associations. Each MAC address then represents a vNIC and a vNIC request is identified by generating a message signaled interrupt (MSI). The number of MAC addresses and interrupts supported by the controller restricts the number of vNICs that can be exported. Although, the finite number of physical resources on the NIC restricts the number of vNICs that can be exported, judicious use of native and para-virtualized access to the vNICs, based on the QoS guarantees a VM needs to honor, can overcome the limitation. A VM that has to support stringent QoS guarantees can choose to use native access to the vNIC whereas those VMs that are looking for best-effort NIC access can be allowed para-virtualized access to a vNIC. The VMM can aid in setting up the appropriate hosting connections based on the requested QoS requirements. The architecture can be realized with the following enhancements:

Virtual-NIC:

In order to define vNIC, the physical device should support timesharing in hardware. For a NIC, this can be achieved by using MSI and dynamically partitionable device memory. These form the basic constructs to define a virtual device on a physical device as depicted in Figure 8. Each virtual device has a specific logical device address, like the MAC address in case of NICs, based on which the MSI is routed. Dedicated DMA channels, a specific set of device registers and a partition of the device memory are part of the virtual device interface which is exported to a VM when it is started. This virtual interface is called the vNIC which forms a restricted address space on the device for the VM to use and remains in possession of the VM until it is active or relinquishes the device. The VMM sets up the device page translation table, mapping the physical device address of the vNIC into the virtual memory of the importing VM, during the vNIC creation and initialization. The device page translation table is given read-only access to the VM and hence forms a significant security provisioning on the device. This prohibits any corrupt device driver of the VM GuestOs to affect other VMs sharing the device or the VMM itself. Also, for high-speed NIC devices, the partitionable memory of the vNIC is useful in setting up large receive and segment offload capabilities specific to each vNIC and thus customize the sizing of each vNIC based on the QoS requirements of the VM.

Accessing virtual-NIC:

To access the vNIC, the native device driver hosted inside the VM replaces the IDD layer. This device driver manipulates the restricted device address space which is exported through the vNIC interface by the VMM. The VMM identifies and forwards the device interrupt to the destination VM. The GuestOS of the VM handles the I/O access and thus directly accounts for the resource usage it incurs. This eliminates the performance interference when the IDD handles multiple VM requests to a shared device. Also, direct access of vNIC to the VM reduces the service time on the I/O accesses. This results in better bandwidth utilization. With the vNIC interface, data transfer is handled by the VM. The VM sets up the Rx/Tx descriptor rings within its address space and makes a request to the VMM for initializing the I/O page translation table during bootup. The device driver uses this table along-with the devices address translation table and does DMA directly into the VM's address space.

QoS and virtual-NIC:

The device memory partition acts as a dedicated device buffer for each of the VMs. With appropriate logic on the NIC card, QoS specific service level agreements (SLAs) can be easily implemented on the device that translate to bandwidth restrictions and VM based processing priority. The key is being able to identify the incoming packet to the corresponding VM. This is done by the NIC based on the packet's associated logical device address. The NIC controller decides on whether to accept or reject the incoming packet based on the bandwidth specification or the current free memory available with the destination vNIC of the packet. This gives a fine-grained control on the incoming traffic and helps reduce the interference effects. The outbound traffic can be controlled by the VM itself, as is done in the existing architectures.

Security and virtual-NIC:

Each vNIC is carved out as a device partition, based on the device requirement specification of the VM. By using appropriate micro-architecture and hardware constructs it can be ensured that a VM does not monopolize device usage and cause denial of service attack to other VMs sharing the device. The architecture allows for unmodified GuestOS on a VM. Hence the security is verified and built outside the VM, i.e. within the VMM. Allowing native device driver within the VM for the vNIC, not only enhances the performance but also allows for easy trapping of the device driver errors by the VMM. This enables for building robust recovery mechanisms for the VM. The model also eliminates sharing of the device access path by allowing direct access to the vNIC by the VM and thereby eliminates the associated failures [36].

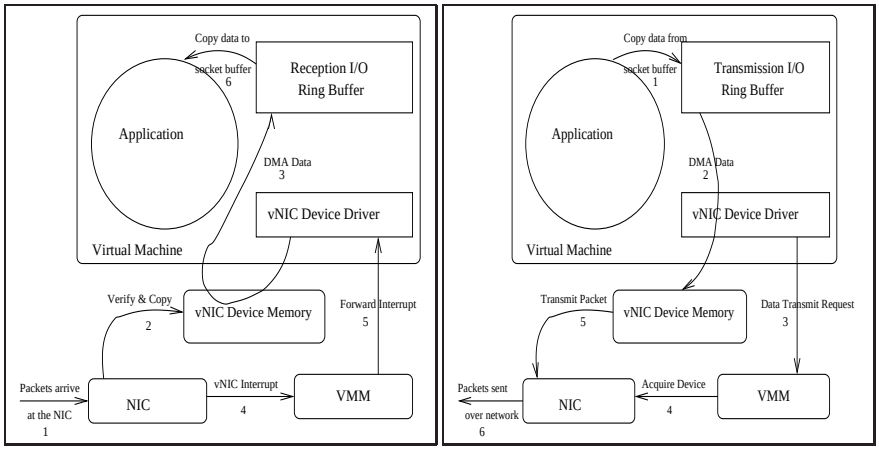
With these constructs, the virtualized NIC is now enabled for carving out secure, customized vNICs for each VM, based on its QoS requirements, and support native device access to the GuestOS of the VM.

5.2 Network Packet work-flow using the virtualized I/O architecture

With the proposed I/O device virtualization architecture, each VM gets safe, direct access to the shared I/O device without having to route the request through the IDD. Only the device interrupts are routed through the VMM. In Figure 9a, and 9b the work-flow for network data reception and transmission using the described device virtualization architecture is depicted. When a packet arrives at the NIC, it deciphers the destination address of the packet, checks if it is a valid destination, then copies the packet into the vNIC's portion of the device memory and issues DMA request to the destination VM based on the vNIC's priority. On completion of the DMA request, the device raises an interrupt. The VMM intercepts the interrupt, determines the destination VM, forwards the interrupt to the VM. The VM's device driver then receives the data from the VM specific device descriptor rings as it would do in the case of non-virtualized server. In the case of transmission, the device driver that is resident in GuestOS of the VM, does a DMA transfer, of the data, directly into the vNIC's mapped memory and sets the appropriate registers to initiate data transmission. The NIC transmits this data based on the vNICs properties like speed, bandwidth and priority. It may be worthwhile to note here that the code changes to support this architecture in the existing implementation will be minimal. Each VM can use the native device driver for it's vNIC. This device driver is the standard device driver for the IOV complaint devices with the only difference that it can only access restricted device address. The device access restrictions in terms of memory, DMA channels, interrupt line and device register sets are setup by the VMM when the VM requests for a virtual device. With the virtual device interface, the VMM now only has to implement the virtual device interrupts.

6 Evaluation of Proposed Architecture

Since the architecture involves the design of a new NIC and a change in both VMM and the device handling code inside the VM's GuestOS, evaluation of the architecture is carried out using simulation based on LQN model of the architecture. In LQN models functional components of the architecture work-flow are represented as server entries. Service of each entry is rendered on a resource. End-to-end work-flow is enacted using entry interactions. The LQN models capture the contention at the resource or software component using service queues. The reason for choosing LQN based modeling is twofold. One, there is a lack of appropriate system simu-



(a) Packet reception work-flow (b) Packet transmission work-flow

Fig. 9: Work-flow of network I/O communication with improvised I/O device virtualization architecture.

lation tools that allow incorporating design of new hardware along-with VMM and GuestOS changes. Second, LQN models are intuitive queuing models that enable capturing of the device and software contention and associated serialization in the end-to-end work-flow, right from the application to the device including the intermediate layers of the VM, IDD and VMM. With appropriate profiling tools, the LQN models are fairly easy to build and are effective in capturing the causes of bottlenecks in the access path. For complete details on general description of LQN modeling and simulation the reader may refer to [3, 4, 5].

6.1 LQN model for the proposed architecture

LQN models can be generated based on the network packet receive and transmit work-flows, manually, using the LQNDEF [3] software developed at the RADS laboratory of *Carleton University*. In the chapter, results generated for the LQN model corressponding to the *httperf* benchmark are presented for analysis, since the bottleneck issues are prominent for this benchmark. For complete details on the generation of the LQN models for the *httperf* benchmark and validation of the models against experimental data, readers may refer to [30, 31]. Three assumptions are made while generating the LQN models used for this analysis, namely:

- The service times established at each of the entries constituting the LQN model are populated based on the service times measured for an *http* request, instead of a *TCP* packet. While it is feasible to model packet level contention, the reason

for choosing request level contention was to enable measurement of the model throughput in terms of the number of satisfied requests. The model validation results demonstrate that there is no significant loss or gain ($<1\%$) of throughput because of this.

- The experimental results for *httperf* benchmark illustrated in section 3 are carried out with varying request rates for a single specified file. In this mode of execution, the file that is fetched as a reply to each of the *http* request, remains constant. Hence the measured service time to process each request remains constant. Also, for the chosen mode of execution of the *httperf* benchmark, the arrival request rate is observed to be uniform. Hence, the service times and arrival rates populated on the LQN model are modeled as deterministic.
- The service time for all device activities that are assumed to be executed in hardware, in the proposed architecture and modeled as separate entities in the LQN model, is set to be significantly low (10^{-10} seconds). For the rest of the software entries, the service times are derived based on the measurements made for the non-virtualized servers. This is justified since the proposed architecture gives native access to the device from within the VM which is assumed to be running the same GuestOS as is used for the non-virtualized server.

In general it is observed that the maximum throughput observed using the LQN model is higher than the experimental observations. The reason for this is simple. For every packet received or transmitted in *Linux*, there are several layers of the network stack that each packet has to pass through. The time taken to traverse this passage is recorded by the profiler as the service time. In the real system, to match the difference between the device speed and CPU speed, appropriate memory buffers (TCP transmit and receive buffers of *Linux* kernel) are maintained. The sizing of these buffers affects the observed application throughput. Observed throughputs are higher for larger buffer sizes. This trend is maintained to the point until the device can handle the rate of network traffic. Once device saturation occurs, the failure behavior usually results in a sudden drop in application throughput. While setting up the LQN model the maximum permissible default buffer size was used in the simulator (which is more than 3 times than what was set on the experimental system). This is normally the adopted practice since in throughput studies the interest is to understand the limits of the model for those service times that make the contention predominant. This gives an idea on the upper-bound of application throughput on a system with maximum possible resources for the service times possible within the desired architecture. The basic idea is to eliminate buffer size constraint in the simulation environment. While it is true that for the proposed architecture in which native access to the I/O device is provided, the maximum throughput that can be achieved, in reality, cannot exceed that of the maximum throughput achieved in the case of non-virtualized server, the results observed using simulations are contradictory. This is because in the simulation environment, the buffer sizes used were much larger than the experimental system. Hence, to make the comparison fair, normalization of simulation results for existing architecture is carried out. To normalize, the LQN model of existing *Xen* architecture is built and simulation results are generated. These results are verified and validated for correctness with that of observed

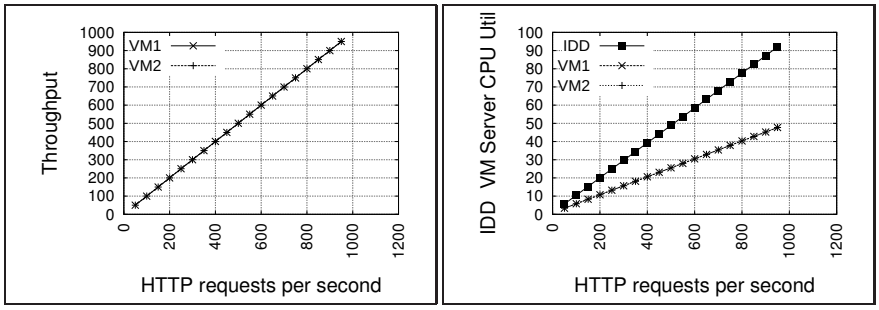
experimental results. After this, all comparisons for the proposed architecture are made using the simulation results of the existing architecture rather than the experimental results.

7 Simulation and Results

The proposed architecture is evaluated using the *parasrvn* simulator of the *LQNS* software package [3].The architecture is evaluated for multicore virtualized servers since the illustrated device sharing dynamics are expected to be pertinent to such systems. The LQN model built for this study consists of one VMM and two VMs and each is pinned to an independent core. In order to compare the performance of the proposed end-to-end architecture within the simulation environment, validation of the LQN model for the existing *Xen* architecture for a multicore server is carried out. Figure 10 depicts the results of achievable throughput and server CPU utilization for a multicore *Xen* server with two VMs consolidated. The throughput graph for both the VMs is similar and appears overlapped in the chart. As it can be noted from Figures 10a and 10b, in a multicore environment with *Xen-IDD*, VM1 and VM2 each pinned to a core, and each VM servicing one *httperf* stream, the maximum throughput, without loss, achievable per stream is 950requests/s as against, 450requests/s in the case of single-core. But, for the maximum throughput, it is observed that the *Xen-IDD*, which is hosting the NIC of the server, the CPU utilization saturates. This indicates that further increase in application throughput is impossible since the processor core serving the *Xen-IDD* has no computing power left. Figure11 shows these statistics for a similar situation but with the proposed I/O virtualization architecture. As one can observe from Figures 11a and 11b, the maximum throughput achievable now per VM increases to 1500 requests/s. This is an increase of application throughput by about 60%. The total throughput achievable at the NIC, derived from consolidating the throughput of both the VMs, also increases by 60% when in comparison to what was achieved on the existing *Xen* architecture.

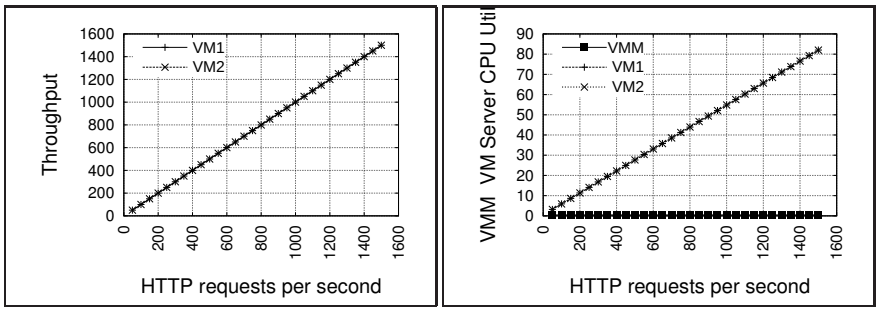
Also, from Figure 11b it is observed that the CPU Utilization of the *IDD* or the hypervisor has considerably reduced and remains bounded by an upper limit. The reason for this behavior is that, the NIC is now handling the identity of the packet destination. Also, in the existing model, bridging software, that routes the packets to a VM and has a substantial overhead, is eliminated in the proposed architecture. The effect is a reduction in the processing time that the *IDD* spends on behalf of each VM. It is also noticed that since the VMM is now spending almost constant time on I/O requests on behalf of the VMs, there is an elimination of performance interference due to varying workloads. This improves the scalability of sharing the device across VMs. With the proposed architecture, each VM is now accountable for all the resource consumption, thereby leading to better QoS controls.

The next evaluation of the proposed architecture is for QoS controls on the network bandwidth. Since the architecture is implemented using LQN model, certain modeling assumptions are made to simulate the network bandwidth controls as im-



(a) Throughput (b) CPU Utilization

Fig. 10: Maximum throughput achievable per *httperf* stream and CPU utilization for existing *Xen* architecture on a multicore server hosting two VMs each servicing one of the *httperf* stream. The IDD, VM1 and VM2 are pinned to independent cores.



(a) Throughput (b) CPU Utilization

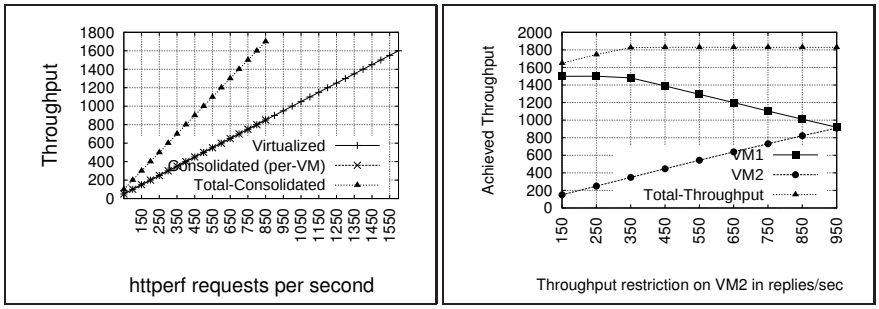
Fig. 11: Maximum achievable throughput and CPU utilization charts for a multicore virtualized server incorporating the proposed I/O virtualization architecture and hosting two VMs, pinned to different cores, each servicing one *httperf* stream.

plemented in the *netfilter* module of *Linux*. LQN model is basically a queuing model wherein, any node (also called entry in *parasrvn* notation) of the queue is described using three parameters, namely, the arrival rate, the service time, and the think time. The arrival rate models the rate of input requests at the entry, service time represents the time the entry takes to process the request before forwarding to the next entry or replying back to the requesting entry and think time denotes the time before which the entry actually services the request. The think time parameter is useful to model policies like bandwidth restrictions, time-sharing intervals, periodic processing, etc. The LQN model is basically a directed acyclic graph that captures the complete work-flow. Hence, the arrival rate is set for the source entry and in this case represents the rate of request arrival at the network interface of the virtualized server.

The service time represents the resource time used for servicing the request by the entry of LQN model and think time is used to model bandwidth restriction. For example, to model 250 requests/second bandwidth restriction, the think time derived is 1/250 seconds. This ensures that the entry will only process 250 requests/second and anything extra will be queued or dropped. The next parameter to model is the burst parameter of the bandwidth control mechanism in *Linux netfilter* module. In *Linux netfilter* module once the bandwidth limit is reached, packet loss occurs. The bandwidth control mechanism also has a burst parameter that allows for some extra packet delivery on the channel, over and above that of imposed bandwidth restriction. By setting the burst rate sufficiently low, equivalent to 10 packets, which is also the minimum that is permissible, it is ensured that the bandwidth control on the constrained channel is tight. The *HTML* page that is requested in the experiments, requires fourteen packets to complete a successful request. Since there is no feature in LQN model to associate the burst parameter of *netfilter*, the QoS experiments were carried out by setting the burst rate to 10 packets. This ensures that for the request, that exceeds the configured bandwidth control, fails and the throughput reported takes into account the desired behavior. Thus, think time setting in LQN model is more restrictive than the *netfilter*. But, since the think time value is based on the deterministic request rate parameter that defines the bandwidth constraint, it still produces equivalent results and this has been validated against observed experimental values [30].

The following graphs in Figure 12 depict the effect of not imposing (Figure 12a) and imposing network bandwidth QoS controls on the incoming stream of VM2 (Figure 12b), in the proposed architecture. The simulations are conducted on a single core server to keep the achievable throughput range within reasonable simulation time. As it can be observed from the graphs of Figure 12a, for the best effort service, the maximum throughput, without loss, achieved by either of the VMs on the consolidated server is equal, indicating a fair share of the resource. The graphs of the Figure 12b show that, unlike as in the case of existing architectures, the QoS constraints when moved to device level, allow the usage of available bandwidth by the unconstrained channel. In the figure, VM2 is constrained to allow requests starting from 150 requests/second to 950 requests/second and VM1 is unconstrained. Since the NIC is discarding requests to VM2 that are above the specified request rate, VM1 can use the available bandwidth, hence higher throughput (1500 replies/sec) on VM1 is achievable. As the bandwidth control on VM2 is relaxed it is noticed that the throughput graphs start converging towards each other and finally merge to that of the best effort case. The bandwidth control on the incoming stream also works to our advantage on the *http* traffic because by discarding the request at the device itself, the server and hence the associated resources, are spared to respond on requests that will eventually be dropped because of bandwidth controls. This control on the device also acts as a strong deterrent for any denial of service type of attacks. The other observation is that when multiple VMs are sharing the NIC, the maximum bandwidth achievable on the unconstrained channel is less ($<$10%) than that which is achieved by the isolated VM. Further reduction on this loss is possible by applying channel based priority and bandwidth control on the outgoing channel

of the constrained VM. The outgoing channel constraints are easily achievable by using existing mechanisms such as those available in the *netfilter* module of *Linux* [35]. The important point to note here is that with faster and higher bandwidth NIC devices, judicious use of large receive and segment offload buffers can lead to higher device utilization without compromising the VM's performance.



(a) Best effort, no QoS on NIC sharing. (b) QoS controls on VM2 outgoing channel.

Fig. 12: Throughput achieved before and after imposing QoS controls on VM2 of the proposed architecture.

8 Conclusion

In this chapter, we described how the lack of virtualization awareness in I/O devices can lead to latency overheads on the I/O path and also cause security vulnerabilities. In addition to this, the intermixing of device management and data protection issues further increases the latency. This results in reducing the effective usable bandwidth of the device. Also, lack of appropriate device sharing control mechanisms, at the device level, leads to loss in bandwidth, causes performance interference on the device sharing VMs and makes the virtualization software the most vulnerable component of the consolidated server. To address these issues I/O device virtualization architecture is proposed. The architecture is an extension to the PCI-SIG IOV specification. The architecture evaluation is done by capturing it as an LQN model and analyzing using simulation of the model. The simulation results show a utilization benefit of about 60%, without enforcing any QoS guarantees or performing any software optimization on the I/O path. The proposed architecture also improves the security and scalability of VMs sharing the NIC. It is demonstrated that by moving the QoS controls to the shared device, the unused bandwidth is made available to the unconstrained VM, unlike the case in prevalent technologies. Although the evaluation is done for para-virtualized systems like *Xen*, it is reasonable to expect that the ideas presented, would benefit fully virtualized systems like *Vmware* since the

architecture enables elimination of the common software entity by providing native device access to the GuestOS of the VM.

Acknowledgements Credits for this work are due to all those unknown reviewers who have meticulously pointed out deficiencies and improvements over several rounds of reviews and also to the summer interns who have enthusiastically carried out the numerous experimental work that helped validate the simulation results.

Appendix

Layered Queuing Network (LQN) Models are the queuing models designed to capture the interdependencies in layered systems. The complete system is described by a set of operations carried out over a set of resources. Every operation requires one or more resources for execution. The LQN model defines an architectural and resource context for each operation. The architectural context defines the initiating event for the operation (execution trigger), when the execution should begin (execution timing) and when it should complete (completion trigger). Based on the semantics of the architectural context, the operation uses resources to carry out its activities, which is defined by its resource context. A resource can be a software entity or a hardware unit involved in actual execution of the operation. Each resource is associated with a queue with a discipline that enforces the order of resource use by the tasks. In layered systems, execution of an activity is carried out by a structured order of operations over resources organized in different layers. An LQN model is necessarily an acyclic graph of all possible sequences of requests to avoid the issue of resource deadlocks. LQNs are very intuitive in capturing resource contentions and thereby the performance implications on a layered system. These models are quite common in practice for modeling software system performance.

The LQN models used in this chapter to evaluate I/O virtualization architecture for the *httperf* benchmark are generated using the software developed at the RADS Laboratory of Carleton University. Complete details of the software, tools and the associated documentation can be found on their website [3].

A short description of the LQN models generated for the proposed I/O virtualization architecture and *Xen* is provided here. The I/O virtualization issues are prominent for the *httperf* benchmark and hence LQN models that capture the end-to-end architecture are generated for analyzing the issues. The diagrams in Figure 13 and Figure 14 depict the LQN models generated for a consolidated Xen server and the proposed I/O virtualization architecture, hosting two VMs. The model has two *httperf* streams accessing *http* servers hosted on different VMs. The model captures the scenario for a multicore system. In these models, each rectangular box represents the conceptual functional entity, that is active in the receive or the transmit path of the network packet work-flows depicted in Figure 9, to complete one *httperf* request-reply sequence. To make the LQN model simpler, few assumptions are made, which are:

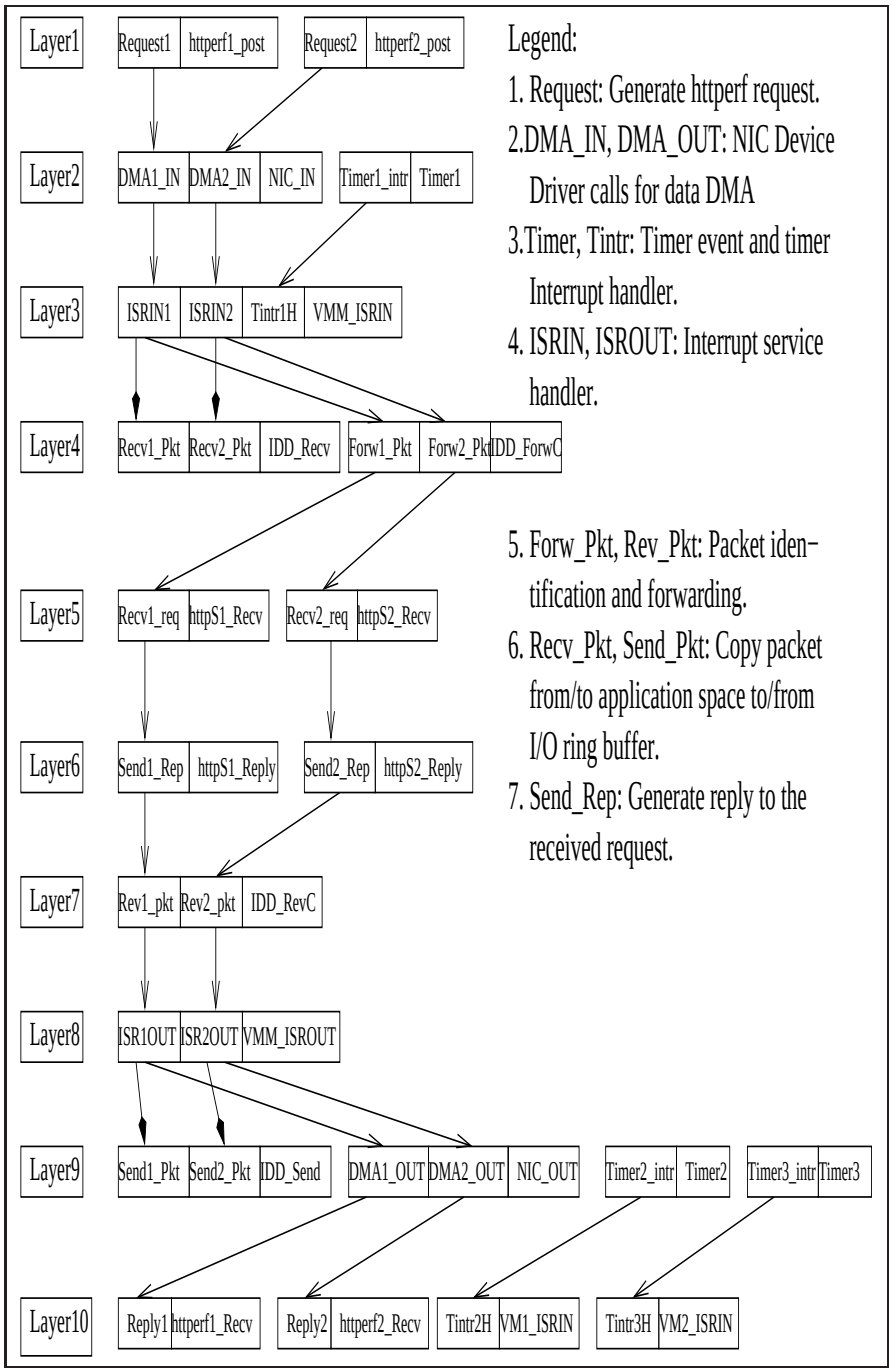


Fig. 13: Layered Queuing Network Model for end-to-end *httperf* benchmark on Xen server.

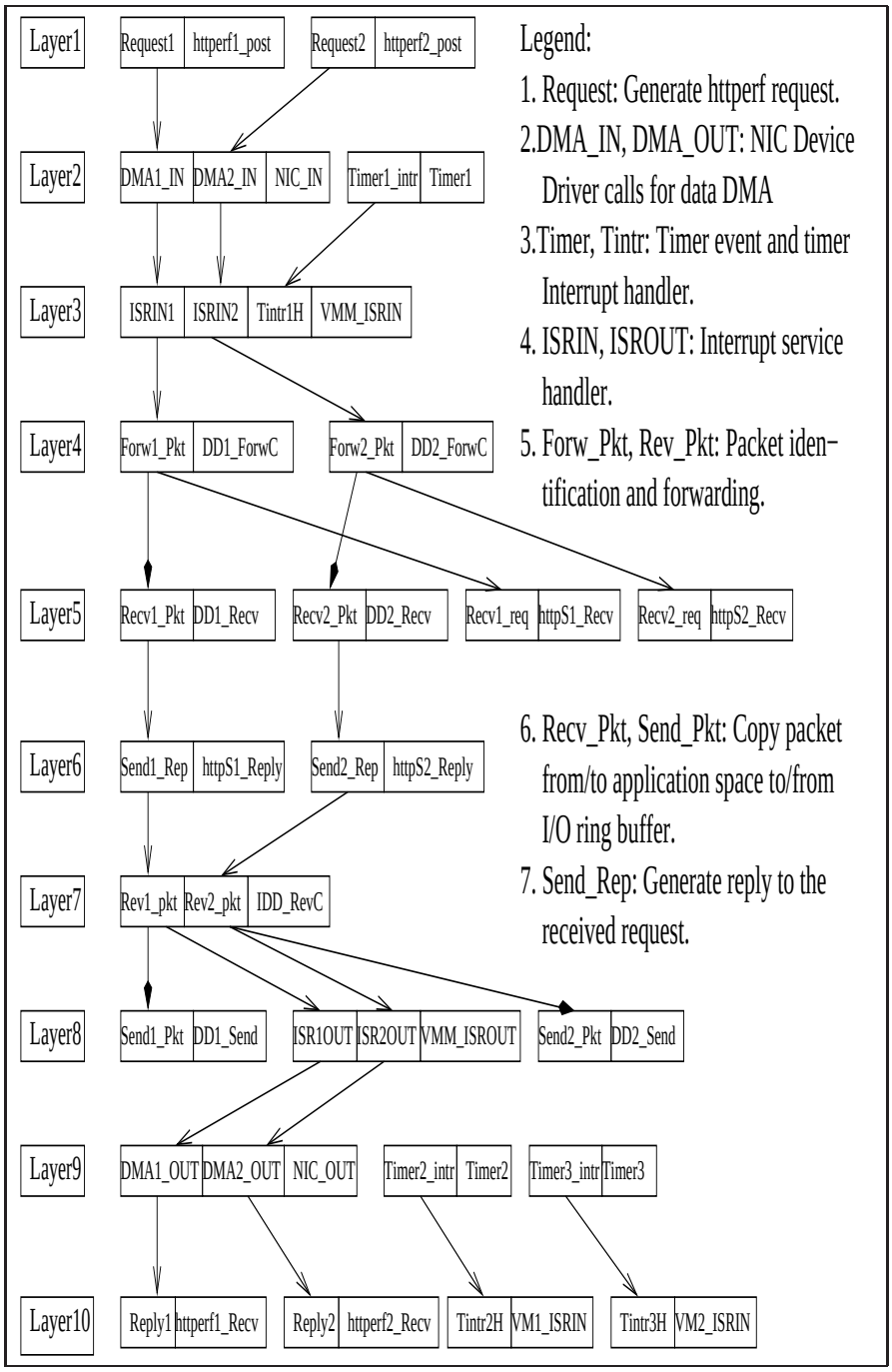


Fig. 14: Layered Queuing Network Model for end-to-end *httperf* benchmark on proposed I/O virtualized server.

1. While in reality every *http* request is broken into a sequence of packets that are passed through the various layers of OS, on an LQN model it is captured as a single service request. This allows for throughput measurements on the model in terms of satisfied *http* requests. This is the unit of measurement for the *httperf* benchmark. By aggregating contention issues from packet level to request level, the throughput measurements tend to be optimistic than what is observed in actual experiments.
2. The service time associated with the transmit/receive operation is consolidated to represent the sending of all the packets composing the *http* request. Because of this assumption the results of the simulation tend to give upper bounds on the achievable throughput when compared to actual implementation. But the deviation is well within 10% of the observed values as reported in [30, 31]. This makes LQN models very useful in evaluating end-to-end architectures.
3. One element that is incorporated in the LQN model and not shown in the workflow is the system timer interrupt using the server element "Timer". This element is introduced in the LQN to account for the queuing delays accrued while the OS is handling timer interrupts. For generating the service time of the interrupt handler, a significantly small delay is used. This value is currently set randomly for want of standard tools to profile kernel procedures.
4. All entries in the LQN model that represent hardware functions are set with a significantly small delay as the service time.

Further details on generating of the LQN models and validating the models against experimental data for this benchmark are discussed in [30, 31].

References

1. Robert P Goldberg, *Survey of Virtual Machine Research*, IEEE-Computer vol7 no 6:34–45, 1974
2. Rick A. Jones, *Netperf: A Network Performance Benchmark Revision 2.0.*, Technical Report, Information Networks Division, Hewlett-Packard Company, 1993. Available online. `http://ci.nii.ac.jp/naid/10000088072/en/.Cited30April2010`
3. RADS Carleton Univ., *Layered Queueing Network Solver software package*, 1995. Available online. `http://www.sce.carleton.ca/rads/lqns.Cited30April2010`
4. J.A. Rolia, K. C. Sevcik, *The Method of Layers*, IEEE Transactions on Software Engineering Vol 21 No 8:689–700, 1995.
5. C. M. Woodside, J. E. Neilson, D. C. Petriu, S. Majumdar, *The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software*, IEEE Transactions on Computers Vol 44 No 1:20–34, 1995.
6. M. Frans Kaashoek, et. Al, *Application Performance and Flexibility on Exokernel Systems*, 16th ACM SOSP:52–65, 1997.
7. Ben Verghese, Anoop Gupta, Mendel Rosenblum, *Performance Isolation: Sharing and Isolation in Shared-Memory Multiprocessors*, ACM SIGPLAN Nov 19:181–192, 1998
8. D. Mosberger, T. Jin, *httperf : A Tool for Measuring Web Server Performance*, ACM Workshop on Internet Server Performance:59–67, 1998
9. T. von Eicken, W. Vogels, *Evolution of the virtual interface architecture*, IEEE Computer Vol 31 No 11:61–68, 1998

10. J. Sugerman, G. Venkatachalam, B. Lim, *Virtualizing I/O devices on VMware Workstation's hosted virtual machine monitor*, Proceedings of the USENIX Annual Technical Conference:1–14, 2001.
11. M. Welsh, D. Culler, *Virtualization considered harmful OS design directions for well-conditioned services*, Hot Topics in OS 8th Workshop:139–144, 2001.
12. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield, *Xen and the art of virtualization*, 19th ACM SIGOPS: 164–177, 2003.
13. K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Wareld, M.Williamson, *Safe hardware access with the Xen virtual machine monitor*, 1st Workshop on OASIS, 2004.
14. *The Globus Resource Specification Language RSL v1.0.*, 2004. Available online. `http://www-fp.globus.org/gram/rsl\_spec1.html.cited30April2010`
15. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, W. Zwaenepoel, *Diagnosing performance overheads in the Xen virtual machine environment*, Proceedings of the ACM/USENIX Conference on Virtual Execution Environments:13–23, 2005.
16. Vmware (2005)*Vmware ESX Server 2 - Architecture and Performance Implications*, 2005. Available online. `http://www.vmware.com/pdf/esx2\_performance\_implications.pdf.Cited30April2010`
17. H. Raj and K. Schwan, *Implementing a scalable selfvirtualizing network interface on a multicore platform*, Workshop on the Interaction between Operating Systems and Computer Architecture, 2005.
18. D. Gupta, L. Cherkasova, R. Gardner, A. Vahdat, *Enforcing performance isolation across virtual machines in Xen* Springer Lecture Notes in Computer Science Vol 4290:342–362, 2006.
19. *Intel Virtualization Technology for Directed-I/O*, 2006. Available online. `www.intel.com/technology/itj/2006/v10i3/2-io/7-conclusion.htm.Cited30April2010`
20. J. Liu, W. Huang, B. Abali, D. K. Panda, *High performance VMMbypass I/O in virtual machines*, Proceedings of the USENIX Annual Technical Conference:3–3, 2006.
21. Menon, A. L. Cox, W. Zwaenepoel, *Optimizing network virtualization in Xen*, Proceedings of the USENIX Annual Technical Conference:2–2, 2006.
22. *PCI-SIG IOV Specification*, 2006. Available online. `http://www.pcisig.com/specifications/iov.Cited30April2010`
23. Santos, J. R., Janakiraman, G., Turner, Y., Pratt, I., *Netchannel 2: Optimizing network performance*, Xen Summit Talk, 2007.
24. Willmann, P., Shafer, J., Carr, D., Menon, A., Rixner, S., Cox, A. L., Zwaenepoel, W. *Concurrent direct network access for virtual machine monitors*, Proceedings of the International Symposium on High-Performance Computer Architecture:306–317, 2007.
25. Kyle J. Nesbit, Miquel Moreto, Francisco J. Cazorla, Alex Ramirez, Mateo Valero, James E. Smith *Multicore Resource Management*, IEEE Micro Special Issue on Interaction of Computer Architecture and Operating System in the Manycore Era vol 28 no 3:6–16, 2008
26. *Neterion*, 2008. Available online. `http://www.neterion.com/.Cited30April2010`
27. *Netxen*, 2008. available online. `http://www.netxen.com/.Cited30april2010`
28. Scott Rixner *Breaking the Performance Barrier: Shared I/O in virtualization platforms has come a long way, but performance concerns remain*, ACM Queue – Virtualization Vol6 No 1:36–ff, 2008.
29. *Sun Microsystems: CrossBow Network Virtualization and Resource Control*, 2008. Available online. `http://www.opensolaris.org/os/community/networking/crossbow\_sunlabs\_ext.pdf.Cited30April2010`

30. J.Lakshmi, S.K.Nandy, *Modeling Architecture-OS interactions using Layered Queuing Network Models*, International Conference Proceedings of HPC Asia:382–389, 2009.
31. J. Lakshmi, S. K. Nandy, *I/O Device virtualization in Multi-core era, a QoS Pespective*, Workshop on Grids, Clouds and Virtualization, Conference on Grids and Pervasive computing:128–135, 2009.
32. Kim, H., Lim, H., Jeong, J., Jo, H., Lee, J., *Task-aware virtual machine scheduling for I/O performance*, Proceedings of ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments:101–110, 2009.
33. Weng, C., Wang, Z., Li, M., Lu, X., *The hybrid scheduling framework for virtual machine systems*, Proceedings of ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments:111–120, 2009.
34. *Solarflare* Communications, 2009. Available online.
`http:://www.solarflare.com/Cited30April2010`
35. *Linux Advanced routing and Traffic contol HowTo*. available online.
`http://lartc.org/howto/index.htmlCited30April2010`
36. J.Lakshmi, S.K.Nandy, *I/O Virtualization Architecture for Security*, To appear in the IEEE Proceedings of International Workshop on Virtualization Technology, 2010.

Architectures for Enhancing Grid Infrastructures with Cloud Computing

Eduardo Huedo, Rafael Moreno-Vozmediano, Rubén S. Montero, and Ignacio M. Llorente

Abstract Grid and Cloud Computing models pursue the same objective of constructing large scale distributed infrastructures, although focusing on complementary aspects. While grid focuses on federating resources and fostering collaboration, cloud focuses on flexibility and on-demand provisioning of virtualized resources. Due to their complementarity, it is clear that both models, or at least some of their concepts and techniques, will coexist and cooperate in existing and future e-infrastructures. This chapter shows how Cloud Computing will help both to overcome many of the barriers to grid adoption, and to enhance the management, functionality, suitability, energy efficiency and utilization of production grid infrastructures.

1 Introduction

Grid infrastructures offer common APIs and service interfaces that make it possible to take advantage of distributed resources without having to modify applications for each site. However, this uniformity unfortunately does not extend to the underlying computing resources, where users are exposed to significant heterogeneities in the computing environment, complicating applications and increasing failure rates.

Eduardo Huedo
Universidad Complutense de Madrid, 28040 Madrid, e-mail: ehuedo@fdi.ucm.es

Rafael Moreno-Vozmediano
Universidad Complutense de Madrid, 28040 Madrid e-mail: rmoreno@dacya.ucm.es

Rubén S. Montero
Universidad Complutense de Madrid, 28040 Madrid e-mail: rubensm@dacya.ucm.es

Ignacio M. Llorente
Universidad Complutense de Madrid, 28040 Madrid e-mail: llorente@dacya.ucm.es

On the other hand, virtualization technologies have matured rapidly over the last few years, providing a mechanism for offering customized, uniform environments for users. This additional flexibility comes with negligible costs in terms of processing power, network bandwidth, and disk I/O in modern systems. Using grid technologies combined with virtualization will allow the grid to provide users with a homogeneous computing environment, simplifying applications and reducing failures.

In parallel with the increasing maturity of virtualization, Cloud Computing technologies have emerged. These technologies allow users to dynamically allocate computing resources and to specify the characteristics for the allocated resources. The fusion of cloud and grid technologies would provide a more dynamic and flexible computing environment for grid application developers.

In computing, a "cloud" usually refers to an "Infrastructure-as-a-Service" (IaaS) cloud, such as Amazon EC2, where IT infrastructure is deployed in the provider's datacenter in the form of Virtual Machines (VM). Cloud computing enables the deployment of an entire IT infrastructure without the associated capital costs, paying only for the used capacity. This new resource provisioning paradigm has been introduced to better respond to changing computing demands, allowing the increase or decrease of capacity in order to meet peak or fluctuating service demands.

With the growing popularity of IaaS clouds, an ecosystem of tools and technologies is emerging that can be used to transform an organization's existing infrastructure into a *private* cloud, so providing a dynamic and flexible private infrastructure to run virtualized service workloads. Private clouds can also support a *hybrid* cloud model by supplementing local infrastructure with computing capacity from an external *public* cloud. A private/hybrid cloud can allow remote access to its resources over the Internet using remote interfaces, such as the web service interfaces used in Amazon EC2, thus making it also a public cloud.

Cloud and virtualization technologies also offer other benefits to administrators of resource centers, such as the migration of live services for load balancing or the deployment of redundant servers. Reduced costs for managing resources immediately benefits users in freeing money for additional computing resources or in having better user support from administrators.

This chapter shows how cloud technology could enhance existing and future grid infrastructures. The structure of this chapter is as follows. Section 2 introduces techniques to enhance grid infrastructures with Cloud Computing. In particular, we envisage the virtualization of grid sites, the delivery of IaaS in grid sites, the cloud scale-out of grid sites, and the federation of grids and clouds. These approaches will be explained in sections 3 to 6. Finally, Section 7 provides some conclusions.

2 Grid Infrastructure Enhancement with Cloud Computing

In the last decade we have witnessed the consolidation of several transcontinental grid infrastructures that have achieved unseen levels of resource sharing. In spite of

this success, current grids suffer from several obstacles that limit their efficiency, namely:

- An increase in the cost and length of the application development and porting cycle. New applications have to be tested in a great variety of environments where the developers have limited configuration capabilities.
- A limitation on the effective number of resources available to each application. Usually, a Virtual Organization (VO) requires a specific software configuration, so an application can be only executed on those sites that support the associated VO. Moreover, the resources devoted to each VO within a site are usually static and cannot be adapted to the VO's workload.
- An increase in the operational cost of the infrastructure. The deployment, maintenance and distribution of different configurations requires specialize, time consuming and error prone procedures. Even worse, new organizations joining a grid infrastructure needs to install and configure a ever-growing middleware stack.

Grid infrastructures can overcome these limitations and can be enhanced in several ways through the use of Cloud Computing concepts. However, we must keep in mind that grid technologies, policies and procedures are the result of many years of research, development and operation. Therefore, we should propose evolutionary, and not revolutionary, steps in the development of better research infrastructures. The approaches we envision are the following:

- Virtualization of grid sites. The integration of private cloud technologies and services, especially virtualization, into existing grid infrastructures would enhance failover and redundancy solutions, and permit machine migration for flexible load balancing and energy efficiency. Virtualization of a grid site would also allow the dynamic provisioning of worker nodes to address the demands of different user communities. This approach will address several needs from resource providers in existing grid infrastructures, being fully transparent to grid application communities, while users would benefit indirectly via the improved stability, reliability, and robustness of the infrastructure.
- IaaS delivery in grid sites. The provision of infrastructure using cloud-like delivery paradigms in addition to existing grid services will address the emerging IaaS cloud-like usage patterns from several user communities. Public cloud interfaces would provide an alternative access to grid site resources to support the execution of any application encapsulated in a VM image. The new interfaces will complement existing grid services, providing a new way to access to the same underlying grid site infrastructure without replacing the grid functionality. In this case, new grid user communities and industrial users would benefit from this innovation in the resource provisioning model of grid sites.
- Cloud scale-out of grid sites. Using hybrid cloud technologies would additionally support "elastic" grid sites able to expand available computing resources in the local cloud to meet peak demands using remote cloud providers. Again, this approach will ease capacity planning for resource providers in existing grid infrastructures, allowing them a quick reaction to peak loads, and still being fully

transparent to grid application communities. Users would benefit indirectly via the on-demand increasing capacity of the infrastructure.

- Federation of grids and clouds. Virtual clusters or grids, as well as individual nodes, can be deployed in public clouds to be accessed from current grid infrastructures using a metascheduler or broker. This way, we would have a federated infrastructure with physical resources statically provisioned (with shared access) from grids complemented with virtual resources dynamically provisioned (with exclusive access) from clouds when needed to meet a given SLA (Service Level Agreement). This technique allows the provision of cloud resources from current grid infrastructures without any change.

The above approaches, based on cloud and virtualization techniques, would provide flexibility, energy efficiency and elasticity to grid sites, and would maximize the utility of grid resources for existing user communities. The first three approaches can be seen as a natural evolution of a grid site, first to become a private cloud, then a public cloud, and finally an hybrid cloud. The fourth approach aims to federate conventional grid sites and cloud resources. These four approaches will be elaborated more in the next sections.

The RESERVOIR[1] and EGEE[2] projects are working together to explore how the institutes providing computing resources to EGEE could benefit from adopting private and hybrid cloud models to provide resources [17]. In particular, the use of a cloud-like provisioning model will allow to easily meet the changing needs of the grid users, from scaling up services to meeting peak loads and improving redundancy or to changing the resources provided to run particular applications. In the context of this collaboration, the StratusLab initiative[3] was created, as an informal collaboration framework, to evaluate the maturity of existing cloud and virtualization technologies and services to enhance production grid infrastructures, and to promote the benefits of virtualization and cloud for the grid community [18].

3 Virtualization of Grid Sites

The pattern of resource demand of the computing community is strongly variable, so making quite difficult to estimate the resource demands. Resource providers need infrastructure solutions, controlled by site administrators, to meet peak demands in their clusters. The usual grid answer is to share between disciplines to smooth out the peaks.

Moreover, in order to be responsive to user requests, the grid must be able to more easily allocate and reallocate its resources. This is currently a problem with the current grid implementation, constrained by its technological choice. Most grid users require a carefully setup environment for their applications to run –e.g. oper-

[1] www.reservoir-fp7.eu

[2] www.eu-egee.org

[3] www.stratuslab.org

ating system, libraries, applications or shared file system– which forces the system administrator to comply with these requirements in order, for a given VO, to successfully run on their resources. In the past, this has also constrained the users to be conservative in their choices of runtime environment, in order to avoid difficult negotiations with the resource owners as time goes on.

In any case, due to the heterogeneity in resource configurations, resources are only useful for a subset of the full user community. Therefore, heterogeneity reduces the opportunities for sharing because underused resources from one community cannot be used to meet peak resource demands from another. Moreover, existing grids suffer from the lack of ability to adapt to the exact requirements of the end-users. Learning from cloud technologies and virtualization, we can now consider a new mode of operation, which completely removes the point of friction between users, looking for a fully customized environment, and administrator, looking for a fully homogeneous one.

Several alternatives have been explored in the past to solve this. For example, the SoftEnv project is a software environment configuration system that allows the users to define the applications and libraries they need [29]. Another common solution is the use of a custom software stack on top of the existing middleware layer, usually referred as pilot-jobs. For example, MyCluster creates a Condor or Sun Grid Engine (SGE) cluster on top of TeraGrid services [31]; and similarly over other middleware we may cite DIRAC [30], glideinWMS [28] or PanDa[22]. Additionally, several projects have investigated the partitioning of a distributed infrastructure to dynamically provide customized independent clusters. For example, COD (Cluster On Demand) is a cluster management software, which dynamically allocates servers from a common pool to multiple virtual clusters [6].

However, the most promising technology to provide VOs with custom execution environments is virtualization. The dramatic performance improvements in hypervisor technologies made possible to experiment with VMs as basic building blocks for computational platforms. In fact, several studies reveal that the virtualization layer has no significant impact on the performance of memory and CPU-intensive applications for HPC clusters [33, 32] or grids[12].

Virtualization offers an attractive solution since it completely separates the host machine (under the control of the system administrators of the resource provider) and the VM running the user operating system. Translated to grid sites, it means that the system administrators remain in full control of their infrastructure, following their own update and upgrade schedule, and remaining free to setup their resource environment as they see fit, as long as they of course comply with the minimum requirements for running VMs. Meanwhile, grid users are now free to compose their VMs exactly as they want it. While there are a few constraints on the VMs depending on the virtualization technology supported, these remain minor compared to the significant benefit that virtualization offers.

The first works in this area integrated resource management systems with VM to provide custom execution environments on a per-job basis. For example Dynamic Virtual Clustering and XGE for MOAB and SGE job managers, respectively [8, 10]. These approaches only overcome the configuration limitation of physical resources

because VMs are bounded to a given resource and only exist during job execution. A similar approach has been implemented at grid level using the GridWay Metascheduler [26]. GridWay[4] allows the definition of an optional phase before the actual execution phase to perform advanced job configuration routines. In this phase, a user defined program (pre-wrapper), executed on the cluster front-end, checks the availability of the requested VM image in the cluster node, transferring it from a GridFTP repository if needed. Then, in the execution phase, another program (wrapper) is executed on a worker node of the cluster. This program starts or restores the VM and waits for its activation by periodically probing its services. When the VM is ready, the program copies all the input files needed to the VM, and executes the user program. When this program exits, output files are copied to the client host and the VM is shut down (or suspended to disk to be recovered later). This strategy does not require additional middleware to be deployed and is not tied to a given virtualization technology. However, since the underlying local resource management system is not aware of the nature of the job itself, some of the potential benefits offered by the virtualization technology (e.g. server consolidation) are not fully exploited.

More general approaches involve the use of VMs as workload units, which implies the change in paradigm from building grids out of physical resources to virtualized ones. For example, the VIOLIN project proposes a novel alternative to application-level overlays based on virtual and isolated networks created on top of an overlay infrastructure. Also, the VMPlant service provides the automated configuration and creation of VMs that can be subsequently be cloned and instantiated to provide homogeneous execution environments across distributed grid resources [15]. On the other hand, the In-VIGO project adds some virtualization layers to the classical grid model, to enable the creation of dynamic pools of virtual resources for application-specific grid-computing [1]. Finally, several studies have explored the use of VMs to provide custom (VO-specific) cluster environments for grid computing. In this case, the clusters are usually completely build up of virtualized resources, as in the Globus Nimbus project [11], or the Virtual Organization Clusters (VOC) [21].

The virtualization of a grid site using private cloud technologies would enable it to meet the changing needs of the users, by adapting and customizing the infrastructure to offer the services required by different application communities. This would allow the grid infrastructure to maximize the utility of the resources, better supporting sharing of resources between communities. For this to work, the right VM has to be instantiated to run a given grid job. Since the cloud API already supports the ability to dynamically instantiate a number of VMs, from an existing store of virtual images, it is possible to manage user-defined virtual images, once the grid API allows such information to be transmitted from the user to the resource manager.

Previous works also highlight that the use of virtualization in grid environments can greatly improve the efficiency, flexibility and sustainability of current productions grids. For example, private cloud technologies would also help sites to dynamically consolidate grid services on a lower number of physical resources, reducing

[4] www.gridway.org

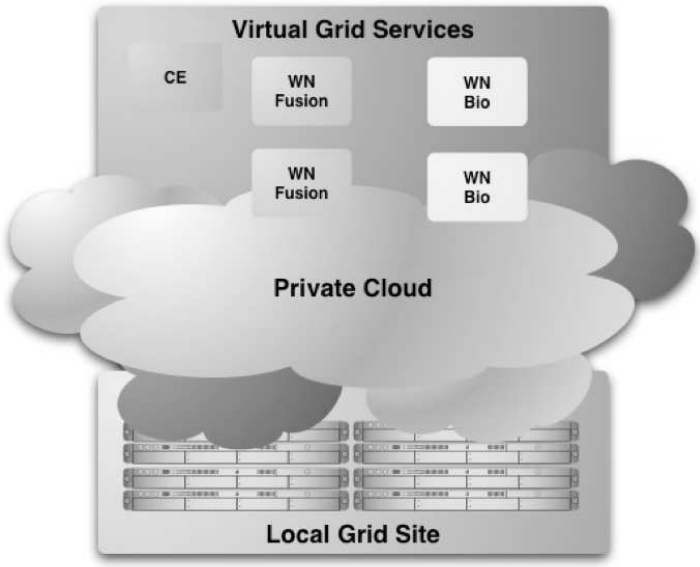


Fig. 1: Virtualization of a grid site to address quick deployment of applications.

the number of active physical systems and thus the administrative effort, power, and cooling required. It is worth noting that energy efficiency is a critical issue for research infrastructures today [3].

Summing up, grids can take advantage of virtualization, not only by extending the classical benefits of VMs for constructing cluster, e.g. consolidation or rapid provisioning of resources [23, 9, 20]; but also by obtaining grid-specific benefits, e.g. support to multiple VOs, isolation of workloads and the encapsulation of services [2].

4 IaaS Delivery in Grid Sites

Many resource providers are interested in using their physical infrastructure to perform other tasks apart from grid service execution, e.g. development of new codes, teaching, creating an internal computing cluster, etc. Moreover, they would like to decide the fraction of resources available via the grid. Resource providers need administrative solutions to easily partition and isolate different clustered services running on the same infrastructure. In fact some resource providers are deploying private cloud facilities in order to support various activities, the grid infrastructure being one of them. The virtualization of the grid site using private cloud technologies will allow organizations to execute multiple virtualized clustered services on

the same physical cluster, dynamically allocating different capacity to the services. Because the same physical infrastructure could be shared by different services, private cloud computing will increase the number of resources provided by existing grid sites, and reduce administration effort.

Once a grid site is virtualized, as explained in the previous section, the provision of infrastructure cloud interfaces would provide an alternate, complementary access to grid site resources and would support the execution of any application encapsulated in a VM image. Using virtualization would increase the number of users by making the hardware useful to a wider range of applications. Moreover, this new functionality would reduce the required manpower in application porting and would attract the science user communities and industrial users that have embraced the cloud computing provisioning model.

This will allow grid sites to turn their site into a public cloud. This way, current and future sites will be equipped with a much more powerful and flexible mean of giving access to their resources to a wider range of users, while not compromising important aspects in data centre managements, such as security, traceability and auditing.

It is important that this cloud API is introduced in a harmonious way with respect to current grid APIs. It is also important that current users of the grid find a sensible migration path to this new way of accessing resources. At the same time, it is equally important that new users find this cloud API to grid resources as convenient and straightforward as possible. In other words, this is an opportunity to streamline the usage of the grid.

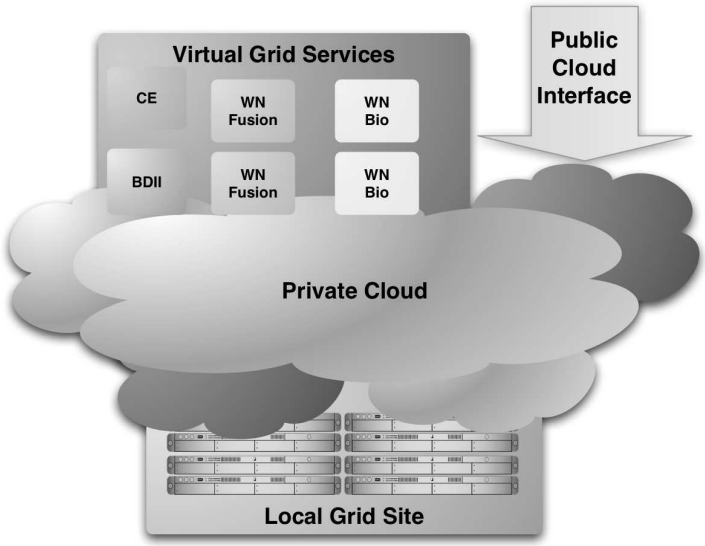


Fig. 2: Offering Infrastructure as a Service.

Having an infrastructure that combines both technologies allows it to serve the maximum number of users, including traditional grid users with computational resources to federate, as we will show in Section 6, and potential new communities that have financial resources to pay for resource utilization, but no resources of their own. This flexibility allows users to structure their applications in a way that is most efficient for them without having to deploy separate resources for each type of infrastructure. It would also allow them to take advantage of commercial providers.

The OpenNebula[5] virtual infrastructure manager provides cloud interfaces like EC2 Query and the OCCI (Open Cloud Computing Interface) standard. Therefore, grid sites virtualized using this technology as virtual infrastructure manager are ready to offer IaaS with very little effort.

5 Cloud Scale-Out of Grid Sites

Hybrid cloud technologies provide an additional method to deal with peak loads, enhancing the system administrator's control over the system. Hybrid cloud computing is also the bridge between existing grid infrastructures and new emerging commercial and science infrastructures based on the cloud model.

Few studies have explored this hybrid cloud model. For example, the VioCluster project enables to dynamically adjust the capacity of a computing cluster by sharing resources between peer domains [27]. Also, as we will show later, the OpenNebula allows the creation of clusters combining physical, virtualized and cloud resources [16]. Finally, important work has already taken place by the open StratusLab collaboration to evaluate the feasibility of running an entire grid site in the Amazon cloud [19], dispensing with any local infrastructure.

A grid site can support a VO beyond its physical resource capabilities by scaling-out to an external cloud provider, such as commercial cloud providers. While, in the case of a commercial cloud provider, the site would have to pay for the resources scaled-out, this means that unprecedented flexibility is provided to grid site administrator. However, control must be put in place to avoid abuse.

This possibility could have a significant impact in reducing infrastructure expenditure since resource owners will no longer have to size their infrastructure for the worst case scenario (peak demand) but for average expected demand. This is possible since the peak demand would simply be scaled-out to third party cloud provider. Over time, as grid sites could also offer a cloud API, as we explained in the previous section, they in return might become resource providers allowing other sites to utilize their excess capacity, if any at a give time. The idea here is that, instead of interchanging jobs, grid sites would interchange resources, encapsulated in VMs. Therefore, this model could in time increase the fluidity of resource allocation and improve resource utilization on a global scale.

[5] www.opennebula.org

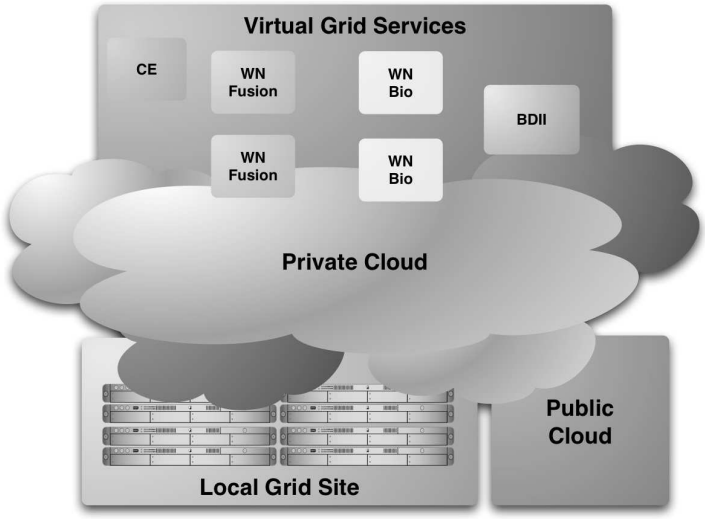


Fig. 3: Combining a grid site with cloud resources to meet a peak demand.

Figure 4 shows a possible implementation of a hybrid cloud using OpenNebula. OpenNebula provides a uniform management layer regardless of the underlying virtualization technology. In this way, OpenNebula can be easily integrated with cloud services by using a specific Amazon EC2 plug-in. The EC2 plug-in then converts the general requests made by OpenNebula core, such as deploy or shutdown, using the EC2 API.

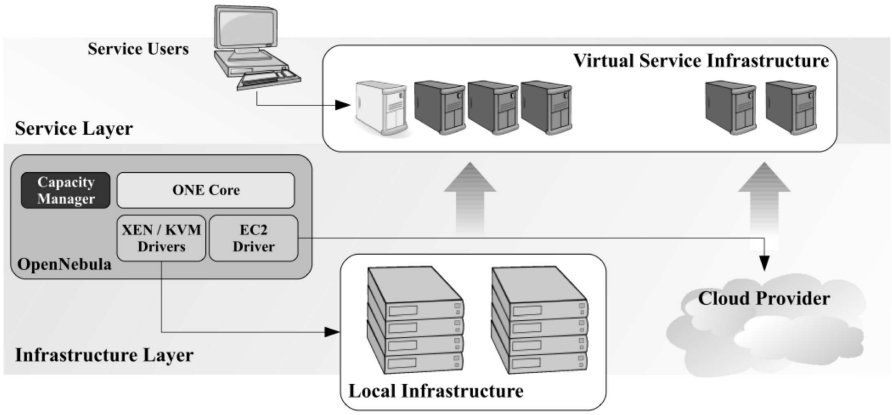


Fig. 4: Hybrid cloud with OpenNebula.

6 Federation of Grids and Clouds

In the recent years there have been a lot of efforts aimed at providing interoperation between grid middlewares to allow the federation of grid infrastructures [24]. This grid interoperation and federation techniques can be also extended to cloud infrastructures.

A set of nodes, possibly grouped in a virtual cluster or a grid site (as explained in the previous section), can be deployed in public clouds to be accessed from current grid infrastructures using a metascheduler or broker, i.e. using Grid Computing concepts, creating a federated infrastructure with physical resources from grids complemented with dynamically provisioned virtual resources from clouds when needed, for example to face peak demands or to meet a given SLA (Service Level Agreement). Grid resources would be shared, while cloud resources would be exclusive, at a given price. This technique allows the provision of cloud resources and their use from current grid infrastructures without any change.

Some architectures have been proposed for this. For example, the InterGrid system uses VMs as building blocks to construct execution environments that span multiple computing sites [7]. Such environments can be created by deploying VMs on different types of resources, like local data centers, grid infrastructures or cloud providers. InterGrid uses OpenNebula as a component for deploying VMs on a local infrastructure. Also, Figure 5 sketches an architecture of a grid infrastructure that can be flexibly built, incorporating new resources temporarily in an automatic fashion to satisfy heavy demands [4]. Moreover, if there is one specific service which is suffering from the peak demand, the system can decide to increase the number of nodes prepared to satisfy such a service. As can be seen, this approach is similar to the cloud scale-out approach presented in the previous section, but now the federation is based on Grid Computing concepts, instead of hybrid Cloud Computing. In addition, we think that this architecture is more natural when using commercial public clouds, since the end user, and not the site administrator, is responsible for the allocation of virtual resources (and will receive the invoice).

One of the building blocks of the architecture presented in Figure 5 is again the GridWay Metascheduler. The flexible architecture of this metascheduler allows the use of adapters (called Middleware Access Drivers, or MADs, in GridWay's terminology), that enable access to different production grid infrastructures [14]. Moreover, it also provides SSH adapters, so access to single nodes can be achieved with decreased overhead, avoiding the need to have installed and configured in the nodes any grid software as, for instance, the Globus Toolkit. Also, the GridWay metascheduler features mechanisms to dynamically discover new resources and it is able to detect and recover from any of the grid elements failure, outage or saturation conditions [13]. Moreover, the metascheduler can handle differences in latencies and performance of resources.

The principal component of the proposed architecture for dynamic provisioning is the Service Manager. This component is used to monitor the GridWay Metascheduler, and when the load of the system excesses a threshold, detected using heuristics, it is responsible to grow the available grid infrastructure using specific adapters

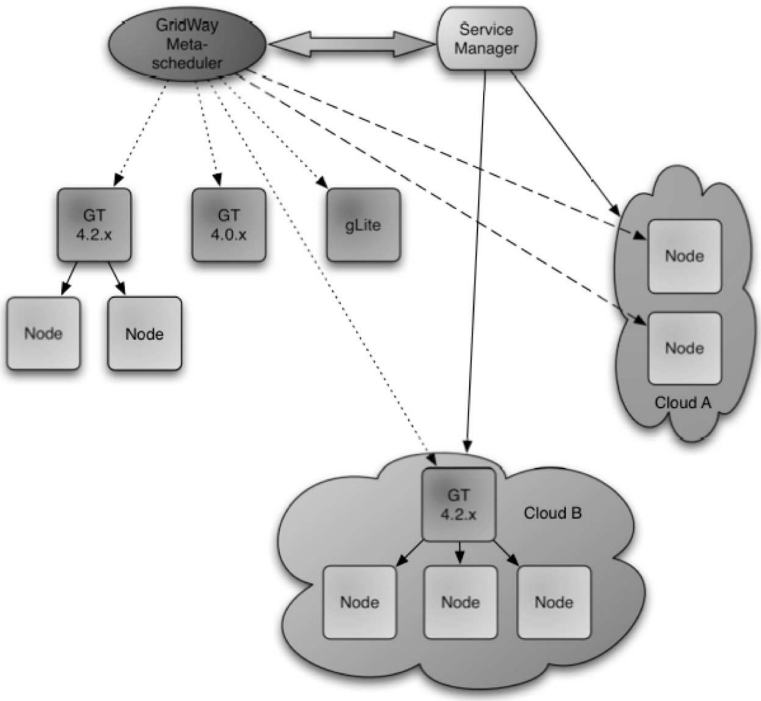


Fig. 5: Federating grids and clouds.

to access different cloud providers. Of course, this component is also responsible to shrink the infrastructure when the load decreases. Clearly, resource provision heuristics should take into account both QoS (Quality of Service) and budget constraints [5] of the end user.

Grid infrastructure growth can be accomplished in two ways. The first one is by requesting a number of single hosts. This corresponds to the use of cloud A in Figure 5. Therefore, this mode adds one single computing resource to the grid infrastructure, that will be accessible through SSH to perform job execution, as GridWay already has such SSH drivers. In this way, machines from cloud providers can be used out-of-the-box, with little to non configuration needed, since basically SSH access is the only requirement.

Another possibility is to deploy a fully virtualized cluster, with a front-end controlling a number of slave nodes. This front-end can then be enrolled to the existing grid infrastructure, adding its capacity [25]. This corresponds to the use of cloud B in the figure. In this second model, negotiation with the cloud provider will grant access to a virtual cluster, accessible through Globus GRAM and controlled by a local resource manager like for example PBS or SGE. This cluster will then be added to the federated grid infrastructure the same way as any other physical sites. Future

work is planned to enrich the flexibility of the grid infrastructure by removing the GRAM layer, enabling GridWay to access the cluster by talking directly to the local resource manager, using the DRMAA (Distributed Resource Management Application API) standard.

7 Conclusions

Cloud Computing has emerged as a very promising paradigm to simplify and improve the management of current IT infrastructures of any kind. Clouds, in their IaaS form, have opened up avenues in this area to ease the maintenance, operation and use of grid sites, and to explore new resource sharing models that could simplify in some cases the porting and development of grid applications. The first works about the joint use of clouds and grids are exploring two main approaches, namely:

- The use of virtualization and cloud techniques as an effective way to provide grid users with custom execution environments. So the same grid site can easily support VOs with different (or even conflicting) configurations. Moreover, grid sites would benefit from improved flexibility, reliability and efficiency.
- The access to grid resources in a cloud-way. So, the users will access "raw" computing capacity by-passing the classical grid middleware stack. This approach is also being considered as a natural way to attract business users to our current e-infrastructures.

The potential benefits that cloud and virtualization technologies can bring to current e-infrastructures require of a common framework that bridge grid and cloud computing models. Various solutions have been proposed to take advantage of these new technologies in a grid environment, from its direct application to encapsulate the execution of each job, to the advance provisioning of virtual clusters.

Grid and cloud technologies address fundamentally different aspects of distributed computing. Grid technology focuses on federation of resources, uniform APIs, common authorization mechanisms, and sharing of resources while cloud computing focuses on dynamic, easy access to resources. Grid site management can be enormously simplified with cloud technologies. But, similarly, cloud resources can be improved by using the common grid authorization mechanisms and movement of files (like VM images) between resources. Because of these complementarities, these technologies will coexist for the foreseeable future and platforms combining them will offer their users a better service.

Acknowledgements This research was supported by Consejería de Educación de la Comunidad de Madrid, Fondo Europeo de Desarrollo Regional (FEDER) and Fondo Social Europeo (FSE), through MEDIANET Research Program S2009/TIC-1468, by Ministerio de Ciencia e Innovación, through the research grant TIN2009-07146, and by the European Union through the StratusLab contract number pending.

References

1. Adabala, S., Chadha, V., Chawla, P., et al.: From virtualized resources to virtual computing grids: the In-VIGO system. Future Generation Computer Systems **21**(6), 896–909 (2005)
2. Begin, M.: An EGEE Comparative Study: Grids and Clouds - Evolution or Revolution. Tech. rep., EGEE-III NA1 (2008). Available at http://edms.cern.ch/file/925013
3. Berl, A., Gelenbe, E., Di Girolamo, M., Giuliani, G., De Meer, H., Dang, M.Q., Pentikousis, K.: Energy-Efficient Cloud Computing. The Computer Journal In press
4. Blanco, C.V., Huedo, E., Montero, R.S., Llorente, I.M.: Dynamic Provision of Computing Resources from Grid Infrastructures and Cloud Providers. In: Proc. Workshop on Grids, Clouds and Virtualization, in conjunction with Grid and Pervasive Computing Conference (GPC 2009), pp. 113–120. IEEE Computer Society (2009)
5. Buyya, R., Murshed, M.M., Abramson, D., Venugopal, S.: Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm. Software – Practice & Experience **35**(5), 491–512 (2005)
6. Chase, J.S., Irwin, D.E., Grit, L.E., Moore, J.D., Sprenkle, S.E.: Dynamic Virtual Clusters in a Grid Site Manager. In: Proc. 12th Intl. Symp. High Performance Distributed Computing (HPDC 2003) (2003)
7. di Costanzo, A., de Assuncao, M., Buyya, R.: Harnessing cloud technologies for a virtualized distributed computing infrastructure. IEEE Internet Computing **13**(5), 24–33 (2009)
8. Emeneker, W., Jackson, D., Butikofer, J., Stanzione, D.: Dynamic Virtual Clustering with Xen and Moab. In: Proc. Frontiers of High Performance Computing and Networking, ISPA 2006 Workshops, *Lecture Notes in Computer Science*, vol. 4331, pp. 440–451 (2006)
9. Emeneker, W., Stanzione, D.: Dynamic virtual clustering. IEEE Cluster (2007)
10. Fallenbeck, N., Picht, H., Smith, M., Freisleben, B.: Xen and the Art of Cluster Scheduling. In: Proc. 1st Intl. Workshop on Virtualization Technology in Distributed Computing (VTDC 2006) (2006)
11. Foster, I., Freeman, T., Keahey, K., Scheftner, D., Sotomayor, B., Zhang, X.: Virtual clusters for grid communities. In: Proc. 6th IEEE Intl. Symp. Cluster Computing and the Grid (CCGrid 2006) (2006)
12. Gilbert, L., Tseng, J., Newman, R., Iqbal, S., Pepper, R., Celebioglu, O., Hsieh, J., Mashayekhi, V., Cobban, M.: Implications of virtualization on grids for high energy physics applications. J. Parallel and Distributed Computing **66**(7), 922–930 (2006)
13. Huedo, E., Montero, R.S., Llorente, I.M.: The GridWay Framework for Adaptive Scheduling and Execution on Grids. Scalable Computing: Practice and Experience **6**, 1–8 (2005)
14. Huedo, E., Montero, R.S., Llorente, I.M.: A Modular Meta-Scheduling Architecture for Interfacing with Pre-WS and WS Grid Resource Management Services. Future Generation Computer Systems **23**(2), 252–261 (2007)
15. Krsul, I., Ganguly, A., Zhang, J., Fortes, J.A.B., Figueiredo, R.J.: VM-Plants: Providing and managing virtual machine execution environments for grid computing. In: Proc. 2004 ACM/IEEE Conference on Supercomputing (2004)
16. Llorente, I.M., Moreno-Vozmediano, R., Montero, R.S.: Cloud Computing for On-Demand Grid Resource Provisioning. In: Proc. High Performance Computing workshop 2008, High Speed and Large Scale Scientific Computing, *Advances in Parallel Computing*, vol. 18, pp. 177–191. IOS Press (2009)
17. Llorente, I.M., Newhouse, S.: Collaboration between the EGEE and RESERVOIR projects. In: EGEE 2009 Conf. (2009)
18. Loomis, C.: StratusLab - Enhancing Grid Infrastructures with Cloud Computing. In: EGEE 2009 Conf. (2009)
19. Loomis, C., Begin, M., Floros, V., Llorente, I.M., Montero, R.S.: Operating a Grid Site in the Cloud. In: 4th EGEE User Forum/OGF 25 and OGF Europe's 2nd International Event (2009)
20. Moreno-Vozmediano, R., Montero, R.S., Llorente, I.M.: Elastic management of cluster-based services in the cloud. In: Proc. 1st Workshop on Automated Control for Datacenters and Clouds (ACDC 2009), in conjunction with 6th Intl. Conf. Autonomic Computing and Communications (ICAC 2009), pp. 19–24. ACM (2009)

21. Murphy, M., Kagey, B., Fenn, M., Goasguen, S.: Dynamic Provisioning of Virtual Organization Clusters. In: Proc. 9th IEEE Intl. Symp. Cluster Computing and the Grid (CCGrid 2009) (2009)
22. Nilsson, P.: Experience from a pilot based system for ATLAS. Journal of Physics: Conference Series **119**(6), 062,038 (2008)
23. Nishimura, H., Maruyama, N., Matsuoka, S.: Virtual clusters on the fly - fast, scalable, and flexible installation. In: Proc. 7th IEEE Intl. Symp. Cluster Computing and the Grid (CCGRID 2007) (2007)
24. Riedel, M., Laure, E., et al.: Interoperation of world-wide production e-Science infrastructures. Concurrency and Computation: Practice and Experience **21**(8), 961–990 (2009)
25. Rodriguez, M., Tapiador, D., Fontan, J., Huedo, E., Montero, R.S., Llorente, I.M.: Dynamic Provisioning of Virtual Clusters for Grid Computing. In: Proc. 3rd Workshop on Virtualization in High-Performance Cluster and Grid Computing (VHPC 2008), in conjunction with Euro-Par 2008, *Lecture Notes in Computer Science*, vol. 5415, pp. 23–32 (2009)
26. Rubio-Montero, A., Huedo, E., Montero, R., Llorente, I.: Management of Virtual Machines on Globus Grids Using GridWay. In: High Performance Grid Computing Workshop (HPGC 2007), in conjunction with 21th Intl. Parallel and Distributed Processing Symp. (IPDPS 2007), pp. 1 –7 (2007)
27. Ruth, P., McGachey, P., Xu, D.: Viocluster: Virtualization for dynamic computational domains. In: 2005 IEEE Intl. Conf. Cluster Computing (2005)
28. Sfiligoi, I.: glideinWMS – A generic pilot-based workload management system. Journal of Physics: Conference Series **119**(6), 062,044 (2008)
29. Teragrid User Support: Managing Your Software Environment. Available at https://www.teragrid.org/web/user-support/environment. Accessed April 2010
30. Tsaregorodtsev, A., Garonne, V., Stokes-Rees, I.: DIRAC: A Scalable Lightweight Architecture for High Throughput Computing. In: Proc. 5th IEEE/ACM Intl. Workshop on Grid Computing (GRID'04), pp. 19–25 (2004)
31. Walker, E., Gardner, J.P., Litvin, V., Turner, E.: Creating Personal Adaptive Clusters for Managing Scientific Jobs in a Distributed Computing Environment. In: Proc. IEEE Workshop on Challenges of Large Applications in Distributed Environments (CLADE 2006) (2006)
32. Youseff, L., Seymour, K., You, H., Dongarra, J., Wolski, R.: The Impact of Paravirtualized Memory Hierarchy on Linear Algebra Computational Kernels and Software. In: Proc. High Performance Distributed Computing (HPDC) (2008)
33. Youseff, L., Wolski, R., Gorda, B., Krintz, C.: Paravirtualization for HPC Systems. In: Proc. Workshop on XEN in HPC Cluster and Grid Computing Environments (XHPC), in conjunction with Intl. Symp. Parallel and Distributed Processing and Application (ISPA 2006) (2006)

Scientific Workflows in the Cloud

Gideon Juve and Ewa Deelman

Abstract The development of cloud computing has generated significant interest in the scientific computing community. In this chapter we consider the impact of cloud computing on scientific workflow applications. We examine the benefits and drawbacks of cloud computing for workflows, and argue that the primary benefit of cloud computing is not the economic model it promotes, but rather the technologies it employs and how they enable new features for workflow applications. We describe how clouds can be configured to execute workflow tasks, and present a case study that examines the performance and cost of three typical workflow applications on Amazon EC2. Finally, we identify several areas in which existing clouds can be improved and discuss the future of workflows in the cloud.

1 Introduction

In this chapter we consider the use of cloud computing for scientific workflow applications. Workflows are coarse-grained parallel applications that consist of a series of computational tasks logically connected by data- and control-flow dependencies. They are used to combine several different computational processes into a single coherent whole. Many different types of scientific analysis can be easily expressed as workflows and, as a result, they are commonly used to model computations in many science disciplines [13]. Using workflow technologies, components developed by different scientists, at different times, for different domains can be used together. Scientific workflows are used for simulation, data analysis, image processing, and many other functions.

Gideon Juve
University of Southern California, Marina del Rey, CA e-mail: juve@usc.edu

Ewa Deelman
University of Southern California, Marina del Rey, CA e-mail: deelman@isi.edu

Scientific workflows can range in size from just a few tasks to millions of tasks. For large workflows it is often desirable to distribute the tasks across many computers in order to complete the work in a reasonable time. As such, workflows often involve distributed computing on clusters, grids [18], and other computational infrastructures. Recently cloud infrastructures are also being evaluated as an execution platform for workflows [24, 28].

Cloud computing represents a new way of thinking about how to deploy and execute scientific workflows. On the one hand, clouds can be thought of as just another platform for executing workflow applications. They support all of the same techniques for workflow management and execution that have been developed for clusters and grids. With very little effort a scientist can deploy a workflow execution environment that mimics the environment they would use on a local cluster or national grid. On the other hand, clouds also provide several features, such as virtualization, that offer new opportunities for making workflow applications easier to deploy, manage and execute. In this chapter we examine those opportunities and describe how workflows can be deployed in the cloud today.

Many different types of clouds are being developed, both as commercial ventures and in the academic community. For the purposes of this chapter we are primarily interested in Infrastructure as a Service (IaaS) clouds [3] as these are more immediately useable by workflow applications. Other clouds, such as Platform as a Service (PaaS) and Software as a Service (SaaS) clouds, may provide additional benefits for creating, managing and executing workflow-based computations, but currently there is a lack of systems developed in this area and additional research is needed to determine how such systems can be fruitfully combined with workflow technologies.

2 Workflows in the Cloud

There is some disagreement about what is the killer feature of cloud computing. For many, especially those in the business community, the attractiveness of cloud is due to its utility-based computing model—the idea that someone else manages a set of computational resources and users simply pay to access them. The academic community, however, has had utility computing for quite a long time in the form of campus clusters, high-performance computing centers such as the NCSA [37] and the SDSC [44], and national cyberinfrastructure such as the TeraGrid [48] and the Open Science Grid [38]. Although the availability of commercial clouds may have some impact on the economics of large-scale scientific computing, we do not view economics as the fundamental benefit of cloud computing for science. Instead, we think clouds provide a multiplicity of benefits that are more technological in nature, and that these benefits stem, primarily, from the extensive use of service-oriented architectures and virtualization in clouds. In the following sections we discuss several aspects of cloud computing that are of particular benefit to workflow applications.

2.1 Provisioning

In grids and clusters scheduling is based on a best-effort model in which a user specifies their computation (the number of resources and amount of time required) and delegates responsibility for allocating resources and scheduling the computation to a batch scheduler. Requests for resources (jobs) are immediately placed into a queue and serviced in order, when resources become available, according to the policies of the scheduler. As a result, jobs often face long, unpredictable queue times, especially jobs that require large numbers of resources, or have long runtimes. The allocation of resources and binding of jobs to those resources are fundamentally tied together and out of the user's control. This is unfortunate for workflows because often the overheads of scheduling jobs on these platforms are high, and for a workflow containing many tasks the penalty is paid many times, which hurts performance [28].

In clouds the process is reversed. Instead of delegating allocation to the resource manager, the user directly provisions the resources required and schedules their computations using a user-controlled scheduler. This provisioning model is ideal for workflows and other loosely-coupled applications because it enables the application to allocate a resource once and use it to execute many tasks, which reduces the total scheduling overhead and can dramatically improve workflow performance [28, 45, 46, 41]. Although in clusters and grids, pilot job systems, such as Condor glide-ins [13] aim to simulate resource provisioning, they face limitations imposed by the target systems, for example the maximum walltime a job is allowed to run on a resource.

2.2 On-demand

Cloud platforms allocate resources on-demand. Cloud users can request, and expect to obtain, sufficient resources for their needs at any time. This feature of clouds has been called the "illusion of infinite resources". The drawback of this approach is that, unlike best-effort queuing, it does not provide an opportunity to wait. If sufficient resources are not available to service a request immediately, then the request fails.

On-demand provisioning allows workflows to be more opportunistic in their choice of resources. Unlike tightly-coupled applications, which need all their resources up-front and would prefer to wait in a queue to ensure priority, a workflow application can start with only a portion of the total resources desired. The minimum usable resource pool for workflows containing only serial tasks is one processor. With on-demand provisioning a workflow can allocate as many resources as possible and start making progress immediately.

2.3 Elasticity

In addition to provisioning resources on-demand, clouds also allow users to return resources on-demand. This dual capability, called elasticity, is a very useful feature for workflow applications because it enables workflow systems to easily grow and shrink the available resource pool as the needs of the workflow change over time. Common workflow structures such as data distribution and data aggregation can significantly change the amount of parallelism in a workflow over time [7]. These changes lead naturally to situations in which it may be profitable to acquire or release resources to more closely match the needs of the application and ensure that resources are being fully utilized.

2.4 Legacy Applications

Workflow applications frequently consist of a collection of complementary software components developed at different times for different uses by different people. Part of the job of a workflow management system is to weave these heterogeneous components into a single coherent application. Often this must be done without changing the components themselves. In some cases no one wants to modify codes that have been designed and tested many years ago in fear of introducing bugs that may affect the scientific validity of outputs. This can be challenging depending on the environment for which the components were developed and the assumptions made by the developer about the layout of the filesystem. These components are often brittle and require a specific software environment to execute successfully.

Clouds and their use of virtualization technology may make these legacy codes much easier to run. Virtualization enables the environment to be customized to suit the application. Specific operating systems, libraries, and software packages, can be installed, directory structures required by the application can be created, input data can be copied into specific locations, and complex configurations can be constructed. The resulting environment can be bundled up as a virtual machine image and redeployed on a cloud to run the workflow.

2.5 Provenance and Reproducibility

Provenance is the origin and history of an object [8]. In computational science, provenance refers to the storage of metadata about a computation that can be used to answer questions about the origins and derivation of data produced by that computation. As such, provenance is the computational equivalent of a lab scientist's notebook and is a critical component of reproducibility, the cornerstone of experimental science.

Virtualization is a useful feature for provenance because it allows one to capture the exact environment that was used to perform a computation, including all of the software and configuration used in that environment. In previous work [23] we proposed a provenance model for workflow applications in which virtual machine images are a critical component. The idea behind this model is that, if a workflow is executed using a virtual machine, then the VM image can be stored along with the provenance of the workflow. Storing the VM image enables the scientist to answer many important questions about the results of a workflow run such as: What version of the simulation code was used to produce the data? Which library was used? How was the software installed and configured? It also enables the scientist to redeploy the VM image and create exactly the same environment that was used to run the original experiment. This environment could be used to rerun the experiment, or it could be modified to run a new experiment. These capabilities are enabled by the use of virtualization in cloud computing.

3 Deploying Workflows in the Cloud

Workflow Management Systems such as Pegasus WMS [19, 16], plan, execute, and monitor scientific workflows. Pegasus WMS, which consists of the Pegasus mapper [14], the DAGMan execution engine [12], and the Condor *schedd* [19] for task execution, performs several critical functions. It adapts workflows to the target execution environment, it manages task execution on distributed resources, it optimizes workflow performance to reduce time to solution and produce scientific results faster, it provides reliability during execution so that scientists need not manage a potentially large number of failures, and they track data so that it can be easily located and accessed during and after workflow execution.

An approach to running workflows on the cloud it to build a virtual cluster using the cloud resources and schedule workflow tasks to that cluster.

3.1 Virtual Clusters

Scientific workflows require large quantities of compute cycles to process tasks. In the cloud these cycles are provided by virtual machines. To achieve the performance required for large-scale workflows many virtual machine instances must be used simultaneously. These collections of VMs are called “virtual clusters” [10, 17, 32]. The process of deploying and configuring a virtual cluster is known as contextualization [31]. Contextualization involves complex configuration steps that can be tedious and error-prone to perform manually. To automate this process, software such as the Nimbus Context Broker [31] can be used. This software gathers information about the virtual cluster and uses it to generate configuration files and start services on cluster VMs.

3.2 Resource Management

Having a collection of virtual machines is not sufficient to run a workflow. Additional software is needed to bind workflow tasks to resources for execution. The easiest way to do this is to use some off-the-shelf resource management system such as Condor [35], PBS [5] or Sun Grid Engine [20]. In this way the virtual cluster mimics a traditional computational cluster. A typical virtual cluster is composed of a virtual machine that acts as a head node to manage the other machines in the cluster and accept tasks from the workflow management system, and a set of worker nodes that execute tasks. Configuration of the resource manager and registration of worker nodes with the head node is part of the process of contextualization.

3.3 Data Storage

Workflows are loosely-coupled parallel applications that consist of a set of discrete computational tasks logically connected via data- and control-flow dependencies. Unlike tightly-coupled applications in which tasks communicate by sending message via the cluster network, workflow tasks typically communicate using files, where each task produces one or more output files that become input files to other tasks. These files are passed between tasks either through a shared storage system or using some file transfer mechanism.

In order to achieve good performance, these storage systems must scale well to handle data from multiple workflow tasks running in parallel on separate nodes. When running on HPC systems, workflows can usually make use of a high-performance, parallel file system such as Lustre [36], GPFS [43], or Panasas [27]. In the cloud, workflows can either make use of a cloud storage service, or they can deploy their own shared file system. To use a cloud storage service the workflow management system would likely need to change the way it manages data. For example, to use Amazon S3 [1] a workflow task needs to fetch input data from S3 to a local disk, perform its computation, then transfer output data from the local disk back to S3. Making multiple copies in this way can reduce workflow performance. Another alternative would be to deploy a file system in the cloud that could be used by the workflow. For example, in Amazon EC2 an extra VM can be started to host an NFS file system and worker VMs can mount that file system as a local partition. If better performance is needed then several VMs can be started to host a parallel file system such as PVFS [34, 50] or GlusterFS [25].

4 Case Study: Scientific Workflows on Amazon EC2

To illustrate how clouds can be used for workflow applications we present a case study using Amazon's EC2 [1] cloud. EC2 is a commercial cloud that exempli-

fies the IaaS model. It provides computational resources in the form of virtual machine instances, which come in a variety of hardware configurations and are capable of running several different virtualized operating systems. For comparison we ran workflows on both EC2 and NCSA's Abe cluster [37]. Abe is a typical example of the resources available to scientists on the national cyberinfrastructure. Running the workflows on both platforms allows us to compare the performance, features, and cost of a typical cloud environment to that of a typical grid environment. Figure 1 shows the high-level set up.

4.1 Applications Tested

Three workflow applications were chosen for this study: an astronomy application (Montage), a seismology application (Broadband), and a bioinformatics application (Epigenome). These applications were selected because they cover a wide range of science domains and resource requirements.

Montage [6] creates science-grade astronomical image mosaics from data collected using telescopes. The size of a Montage workflow (Figure 2) depends upon the area of the sky (in square degrees) covered by the output mosaic. In our experiments we configured Montage workflows to generate an 8-degree mosaic. The resulting workflow contains 10,429 tasks, reads 4.2 GB of input data, and produces 7.9 GB of output data.

Broadband [9] generates and compares seismograms from several high- and low-frequency earthquake simulation codes. Each Broadband workflow (Figure 3) generates seismograms for several sources (scenario earthquakes) and sites (geographic locations). For each (source, site) combination the workflow runs several high- and low-frequency earthquake simulations and computes intensity measures of the resulting seismograms. In our experiments we used 4 sources and 5 sites to generate a workflow containing 320 tasks that reads 6 GB of input data and writes 160 MB of output data.

Epigenome [30] maps short DNA segments collected using high-throughput gene sequencing machines to a previously constructed reference genome using the MAQ software [33]. The workflow (Figure 4) splits several input segment files into small chunks, reformats and converts the chunks, maps the chunks to a reference genome, merges the mapped sequences into a single output map, and computes the sequence density for each location of interest in the reference genome. The workflow used in our experiments maps human DNA sequences to a reference chromosome 21. The workflow contains 81 tasks, reads 1.8 GB of input data, and produces 300 MB of output data.

4.2 Software

All workflows were planned and executed using the Pegasus WMS. Pegasus is used to transform abstract, resource-independent workflow descriptions into concrete, platform-specific execution plans. These plans are executed using DAGMan to track dependencies and release tasks as they become ready, and Condor schedd to run tasks on the available resources.

4.3 Hardware

Table 1: Resource Types Used

Type	Arch.	CPU	Cores	Memory	Network	Storage	Price
m1.small	32-bit	2.0-2.6 GHz Opteron	1/2	1.7 GB	1-Gbps Ethernet	Local disk	$0.10/hr
m1.large	64-bit	2.0-2.6 GHz Opteron	2	7.5 GB	1-Gbps Ethernet	Local disk	$0.40/hr
m1.xlarge	64-bit	2.0-2.6 GHz Opteron	4	15 GB	1-Gbps Ethernet	Local disk	$0.80/hr
c1.medium	32-bit	2.33-2.66 GHz Xeon	2	1.7 GB	1-Gbps Ethernet	Local disk	$0.20/hr
c1.xlarge	64-bit	2.33-2.66 GHz Xeon	8	7.5 GB	1-Gbps Ethernet	Local disk	$0.80/hr
abe.local	64-bit	2.33 GHz Xeon	8	8 GB	10-Gbps InfiniBand	Local disk	N/A
abe.lustre	64-bit	2.33 GHz Xeon	8	8 GB	10-Gbps InfiniBand	Lustre	N/A

EC2 provides resources with various hardware configurations. Table 1 compares the resource types used for the experiments. It lists five resource types from EC2 (m1.* and c1.*) and two resource types from Abe (abe.local and abe.lustre). There are several noteworthy details about the resources shown. First, although there is actually only one type of Abe node, there are two types listed in the table: abe.local and abe.lustre. The actual hardware used for these types is equivalent; the difference is in how I/O is handled. The abe.local type uses a local partition for I/O, and the abe.lustre type uses a Lustre partition for I/O. Using two different names is simply a notational convenience. Second, in terms of computational capacity, the c1.xlarge resource type is roughly equivalent to the abe.local resource type with the exception that abe.local has slightly more memory. We use this fact to estimate the virtualization overhead for our test applications on EC2. Third, in rare cases EC2 assigns Xeon processors for m1.* instances, but for all of the experiments reported here the m1.* instances used were equipped with Opteron processors. The only significance is that Xeon processors have better floating-point performance than Opteron processors (4 FLOP/cycle vs. 2 FLOP/cycle). Finally, the m1.small instance type is shown having core. This is possible because of virtualization. EC2 nodes are configured to give m1.small instances access to the processor only 50% of the time. This allows a single processor core to be shared equally between two separate m1.small instances.

4.4 Execution Environment

The execution environment was deployed on EC2 as shown in Figure 5 (left). A submit host running outside the cloud was used to coordinate the workflow, and worker nodes were started inside the cloud to execute workflow tasks. For the Abe experiments Globus [47] and Corral [29, 11] were used to deploy Condor glideins [22] as shown in Figure 5(right). The glideins started Condor daemons on the Abe worker nodes, which contacted the submit host and were used to execute workflow tasks. This approach creates an execution environment on Abe that is equivalent to the EC2 environment.

4.5 Storage

Amazon provides several storage services that can be used with EC2. The Simple Storage Service (S3) [39] is an object-based storage service. It supports PUT, GET, and DELETE operations for un-typed binary objects up to 5 GB in size. The Elastic Block Store (EBS) [2] is a block-based storage service that provides SAN-like storage volumes up to 1 TB in size. These volumes appear as standard block devices when attached to an EC2 instance and can be formatted with standard UNIX file systems. EBS volumes cannot be shared between multiple instances.

In comparison, Abe provides shared storage on a large Lustre [36] parallel file system. This is a very common storage configuration for cluster and grid platforms.

To run workflow applications storage must be allocated for application executables, input data, intermediate data, and output data. In a typical workflow application executables are pre-installed on the execution site, input data is copied from an archive to the execution site, and output data is copied from the execution site to an archive.

For all experiments, executables and input data were pre-staged to the execution site, and output data were not transferred from the execution site. For the EC2 experiments, executables were installed in the VM images, input data was stored on EBS volumes, and intermediate and output data was written to a local partition. For the Abe experiments executables and input data were kept on the Lustre file system and intermediate and output data was written in some cases to a local partition (abe.local experiments) and in other cases to the Lustre file system (abe.lustre experiments).

EBS was chosen to store input data for a number of reasons. First, storing inputs in the cloud obviates the need to transfer input data repeatedly. This saves both time and money because transfers cost more than storage. Second, using EBS avoids the 10 GB limit on VM images, which is too small to include the input data for all the applications tested. We can access the data as if it were on a local disk without packaging it in the VM image. A simple experiment using the disk copy utility ‘dd’ showed similar performance reading from EBS volumes and the local disk (74.6 MB/s for local, and 74.2 MB/s for EBS). Finally, using EBS simplifies our setup by allowing us to reuse the same volume for multiple experiments. When we need

to change instances we just detach the volume from one instance and re-attach it to another.

4.6 Performance Comparison

In this section we compare the performance of the selected workflow applications by executing them on Abe and EC2. The critical performance metric we are concerned with is the runtime of the workflow (also known as the makespan), which is the total amount of wall clock time from the moment the first workflow task is submitted until the last task completes. The runtimes reported for EC2 do not include the time required to install and boot the VM, which typically averages between 70 and 90 seconds [26]. Now this is much less if you use EBS to store images., and the runtimes reported for Abe do not include the time glidein jobs spend waiting in the queue, which is highly dependent on the current system load. Also, the runtimes do not include the time required to transfer input and output data (see Table 4). We assume that this time will be variable depending on WAN conditions. A study of bandwidth to/from Amazon services is presented in [39]. In our experiments we typically observed bandwidth on the order of 500-1000KB/s between Amazon's East Region datacenter and our submit host in Marina del Rey, CA.

We estimate the virtualization overhead for each application by comparing the runtime on c1.xlarge with the runtime on abe.local. Measuring the difference in runtime between these two resource types should provide a good estimate of the cost of virtualization.

Figure 6 shows the runtime of the selected applications using the resource types shown in Table 2. In all cases the m1.small resource type had the worst runtime by a large margin. This is not surprising given its relatively low capabilities.

For Montage the best EC2 performance was achieved on the m1.xlarge type. This is likely due to the fact that m1.xlarge has twice as much memory as the next best resource type. The extra memory is used by the Linux kernel for the file system buffer cache to reduce the amount of time the application spends waiting for I/O. This is particularly beneficial for Montage, which is very I/O-intensive. The best overall performance for Montage was achieved using the abe.lustre configuration, which was more than twice as fast as abe.local. This large gap suggests that having a parallel file system is a significant advantage for I/O-bound applications like Montage. The difference in runtime between the c1.xlarge and abe.local experiments suggests that the virtualization overhead for Montage is less than 8%.

The best overall runtime for Broadband was achieved by using the abe.lustre resource type, and the best EC2 runtime was achieved using the c1.xlarge resource type. This is despite the fact that only 6 of the 8 cores on c1.xlarge and abe.lustre could be used due to memory limitations. Unlike Montage the difference between running Broadband on a relatively slow local disk (abe.local) and running on the parallel file system (abe.lustre) is not as significant. This is attributed to the lower

I/O requirements of Broadband. Broadband performs the worst on m1.small and c1.medium, which also have the lowest amount memory (1.7 GB). This is because m1.small has only half a core, and c1.medium can only use one of its two cores because of memory limitations. The difference between the runtime using c1.xlarge and the runtime using abe.local was only about 1%. This small difference suggests a relatively low virtualization penalty for Broadband.

For Epigenome the best EC2 runtime was achieved using c1.xlarge and the best overall runtime was achieved using abe.lustre. The primary factor affecting the performance of Epigenome was the availability of processor cores, with more cores resulting in a lower runtime. This is expected given that Epigenome is almost entirely CPU-bound. The difference between the abe.lustre and abe.local runtimes was only about 2%, which is consistent with the fact that Epigenome has relatively low I/O and is therefore less affected by the parallel file system. The difference between the abe.local and the c1.xlarge runtimes suggest that the virtualization overhead for this application is around 10%, which is higher than both Montage and Broadband. This may suggest that virtualization has a larger impact on CPU-bound applications.

Based on these experiments we believe the performance of workflows on EC2 is reasonable given the resources that can be provisioned. Although the EC2 performance was not as good as the performance on Abe, most of the resources provided by EC2 are also less powerful. In the cases where the resources are similar, the performance was found to comparable. The EC2 c1.xlarge type, which is nearly equivalent to abe.local, delivered performance that was nearly the same as abe.local in our experiments.

For I/O-intensive workflows like Montage, EC2 is at a significant disadvantage because of the lack of high-performance parallel file systems. While such a file system could conceivably be constructed from the raw components available in EC2, the cost of deploying such a system would be prohibitive. In addition, because EC2 uses commodity networking equipment it is unlikely that there would be a significant advantage in shifting I/O from a local partition to a parallel file system across the network, because the bottleneck would simply shift from the disk to the network interface. In order to compete performance-wise with Abe for I/O-intensive applications, Amazon would need to deploy both a parallel file system and a high-speed interconnect.

For memory-intensive applications like Broadband, EC2 can achieve nearly the same performance as Abe as long as there is more than 1 GB of memory per core. If there is less, then some cores must sit idle to prevent the system from running out of memory or swapping. This is not strictly an EC2 problem, the same issue affects Abe as well.

For CPU-intensive applications like Epigenome, EC2 can deliver comparable performance given equivalent resources. The virtualization overhead does not seem to be a significant barrier to performance for such applications. In fact, the virtualization overhead measured for all application less than 10%. This is consistent with previous studies that show similar virtualization overheads [4, 21, 49]. As such, virtualization does not seem, by itself, to be a significant performance problem for

clouds. As virtualization technologies improve it is likely that what little overhead there is will be further reduced or eliminated.

4.7 Cost Analysis

In this section we analyze the cost of running workflow applications in the cloud. We consider three different cost categories: resource cost, storage cost, and transfer cost. Resource cost includes charges for the use of VM instances in EC2; storage cost includes charges for keeping VM images in S3 and input data in EBS; and transfer cost includes charges for moving input data, output data and log files between the submit host and EC2.

4.7.1 Resource Cost

Each of the five resource types Amazon offers is charged at a different hourly rate: $0.10/hr for m1.small, $0.40/hr for m1.large, $0.80/hr for m1.xlarge, $0.20/hr for c1.medium, and $0.80/hr for c1.xlarge. Usage is rounded up to the nearest hour, so any partial hours are charged as full hours.

Figure 7 shows the per-workflow resource cost for the applications tested. Although it did not perform the best in any of our experiments, the most cost-effective instance type was c1.medium, which had the lowest execution cost for all three applications.

4.7.2 Storage Cost

Storage cost consists of a) the cost to store VM images in S3, and b) the cost of storing input data in EBS. Both S3 and EBS use fixed monthly charges for the storage of data, and variable usage charges for accessing the data. The fixed charges are $0.15 per GB-month for S3, and $0.10 per GB-month for EBS. The variable charges are $0.01 per 1,000 PUT operations and $0.01 per 10,000 GET operations for S3, and $0.10 per million I/O operations for EBS. We report the fixed cost per month, and the total variable cost for all experiments performed.

We used a 32-bit and a 64-bit VM image for all of the experiments in this paper. The 32-bit image was 773 MB and the 64-bit image was 729 MB for a total fixed cost of $0.22 per month. In addition, there were 4616 GET operations and 2560 PUT operations for a total variable cost of approximately $0.03.

The fixed monthly cost of storing input data for the three applications is shown in Table 2. In addition, there were 3.18 million I/O operations for a total variable cost of $0.30.

Table 2: Monthly storage cost

Application	Volume Size	Monthly Cost
Montage	5GB	$0.66
Broadband	5GB	$0.60
Epigenome	2GB	$0.26

4.7.3 Transfer Cost

In addition to resource and storage charges, Amazon charges $0.10 per GB for transfer into, and $0.17 per GB for transfer out of, the EC2 cloud. Tables 3 and 4 show the per-workflow transfer sizes and costs for the three applications studied. Input is the amount of input data to the workflow, output is the amount of output data, and logs is the amount of logging data that is recorded for workflow tasks and transferred back to the submit host. The cost of the protocol used by Condor to communicate between the submit host and the workers is not included, but it is estimated to be less than $0.01 per workflow.

Table 3: Per-workflow transfer sizes

Application	Input	Output	Logs
Montage	4291 MB	7970 MB	40 MB
Broadband	4109 MB	159 MB	5.5 MB
Epigenome	1843 MB	299 MB	3.3 MB

Table 4: Per-workflow transfer costs

Application	Input	Output	Logs	Total
Montage	$0.42	$1.32	< $0.01	$1.75
Broadband	$0.40	$0.03	< $0.01	$0.43
Epigenome	$0.18	$0.05	< $0.01	$0.23

The first thing to consider when provisioning resources on EC2 is the tradeoff between performance and cost. In general, EC2 resources obey the aphorism “you get what you pay for”—resources that cost more perform better than resources that cost less. For the applications tested, c1.medium was the most cost-effective resource type even though it did not have the lowest hourly rate, because the type with the lowest rate (m1.small) performed so badly.

Another important thing to consider when using EC2 is the tradeoff between storage cost and transfer cost. Users have the option of either a) transferring input data for each workflow separately, or b) transferring input data once, storing it in the cloud, and using the stored data for multiple workflow runs. The choice of which approach to employ will depend on how many times the data will be used, how long the data will be stored, and how frequently the data will be accessed. In general, storage is more cost-effective for input data that is reused often and accessed frequently, and transfer is more cost-effective if data will be used only once. For the applications tested in this paper, the monthly cost to store input data is only slightly more than the cost to transfer it once. Therefore, for these applications, it is usually more cost-effective to store the input data rather than transfer the data for each workflow.

Although the cost of transferring input data can be easily amortized by storing it in the cloud, the cost of transferring output data may be more difficult to reduce. For many applications the output data is much smaller than the input data, so the cost of transferring it out may not be significant. This is the case for Broadband and Epigenome, for example. For other applications the large size of output data may be cost-prohibitive. In Montage, for example, the output is actually larger than the input and costs nearly as much to transfer as it does to compute. For these applications it may be possible to leave the output in the cloud and perform additional analyses there rather than to transfer it back to the submit host.

In [15] the cost of running 1-, 2-, and 4-degree Montage workflows on EC2 was studied via simulation. That paper found the lowest total cost of a 1-degree workflow to be $0.60, a 2-degree to be $2.25, and a 4-degree to be $9.00. In comparison, we found the total cost of an 8-degree workflow, which is 4 times larger than a 4-degree workflow, to be approximately $1.25 if data is stored for an entire month, and $2.35 if data is transferred. This difference is primarily due to an underestimate of the performance of EC2 that was used in the simulation, which produced much longer simulated runtimes.

Finally, the total cost of all the experiments presented in this paper was $149.55. That includes all charges related to learning to use EC2, creating VM images, and running test and experimental workflows.

5 Challenges

In the experiments above we focused on running a workflow application on a single, multi-core virtual instance. There are several challenges that need to be addressed when running workflows on multiple virtual instances. Here we describe the challenges related to data storage, communications, and configurability.

5.1 Lack of Appropriate Storage Systems

Existing workflow systems often rely on parallel and distributed file systems. These are required to ensure that tasks landing on any node can access the outputs of previous tasks that may have executed on another node. It is possible to transfer inputs and outputs for each task separately, however the repeated movement of data is highly inefficient, and time-consuming. In addition, it may be costly in a commercial cloud that charges by the number of bytes transferred. Commercial clouds often deploy structured or object-based storage services that can be utilized by workflow applications. However, these services typically do not provide standard file system interfaces. In order to use these systems the application codes must either be modified to interface with the storage services, or must be wrapped with additional workflow components that know how to do the translation. Another solution is to deploy a temporary shared file system in the cloud as part of a virtual cluster, but the problems with this solution are that it is complex, potentially costly, and requires an additional step to ensure that desired outputs are transferred to permanent storage. A better solution would be a permanent, scalable, parallel file system similar to what existing clusters and grids use. The challenge to this approach is that it is not clear how such a file system would be provisioned and shared among different users within a cloud. In particular, authentication and authorization would be key challenges of such a system.

5.2 Relatively Slow Networks

Most communication in workflows occurs through intermediate files that are written by one task and read by a subsequent task. In a distributed environment these files need to be transferred across the network in order for the workflow to make progress. As such, workflows depend on high-performance networks to achieve good performance. This is especially true for data-intensive workflows. Networks that provide high-throughput, but not necessarily low latency are ideal, however the predominant networking technology employed by existing commercial clouds is Gigabit Ethernet. In comparison, many cluster and grid infrastructures provide interconnects that deliver 10 Gigabit/second or better. In order for clouds to be a viable alternative to clusters and grids as a platform for workflow applications they would need to deploy much faster networks. Making high-performance networking function with OS-level virtualization should be a top priority for future cloud platforms.

5.3 Lack of Tools

Setting up an environment to run workflows in the cloud is a complex endeavor. There is some work in virtual appliances [42], but those are typically designed for

single nodes and not for clusters of nodes. The Nimbus Context Broker [31] can be used to construct virtual clusters and is immensely useful for running workflows in the cloud. More tools are needed to simplify the management of cloud-based environments.

6 Summary and Future Outlook

The benefit of cloud computing for science is not necessarily in its utility computing and economic aspects, which are not new for academic computing. The benefit of clouds is rather in its technological features that stem from service-oriented architecture and virtualization. In the area of scientific workflows clouds have many important benefits. These benefits include the illusion of infinite resources, lease-based provisioning, elasticity, support for legacy applications and environments, provenance and reproducibility, and others.

In our work, we supported the workflow execution on the cloud through the deployment of a Condor-based virtual cluster on top of virtual instance. Workflows can also be made to run across multiple virtual instances, but additional communicate of data files need to be supported.

Cloud platforms like Amazon EC2 can be used to execute workflows today, but in the future much work is needed to bring these platforms up to the performance level of the grid. This includes developing cloud storage systems that are appropriate for workflow and other science applications as well as tools to help scientists and workflow engineers deploy their applications in the cloud.

In the future we will see many new cloud-based solutions for workflow applications. For example, we will likely see the development of new management tools that help users run workflows using existing tools in the grid. We may see new workflow systems that are designed with the specific features of the cloud in mind. We will see research on how to deploy workflows across grids and clouds. We may see PaaS and SaaS clouds that are developed exclusively for workflow applications. For example, a PaaS cloud may provide a user-centered Replica Location Service (RLS) for locating files in the cloud and outside, a dynamic Network Attached Storage (NAS) service for storing files used and created by workflows, a transformation catalog service to store and manage application binaries, and a workflow execution service for managing tasks and dependencies. A SaaS cloud for workflows may provide an application-specific portal where a user could enter the details of a desired computation and have the underlying workflow services generate a new workflow instance, plan and execute the computation, and provide access to the results.

Acknowledgements We acknowledge the contributions of Karan Vahi, Gaurang Mehta, Phil Maechling, Benjamin P. Berman, and Bruce Berriman. This work was supported by the National Science Foundation under the SciFlow (CCF-0725332) grant. This research made use of Montage, funded by the National Aeronautics and Space Administration's Earth Science Technology Office, Computation Technologies Project, under Cooperative Agreement Number NCC5-626 between NASA and the California Institute of Technology.

References

1. Amazon.com: Amazon web services (aws). http://aws.amazon.com
2. Amazon.com: Elastic block store (ebs). http://aws.amazon.com/ebs
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Tech. rep., UC Berkeley (2009)
4. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proceedings of the 19th ACM symposium on Operating systems principles (2003)
5. Bayucan, A., Henderson, R.L., Lesiak, C., Mann, B., Proett, T., Tweten, D.: Portable batch system: External reference specification. Tech. rep., MRJ Technology Solutions (1999)
6. Berriman, B., Bergou, A., Deelman, E., Good, J., Jacob, J., Katz, D., Kesselman, C., Laity, A., Singh, G., Su, M.H., Williams, R.: Montage: A grid-enabled image mosaic service for the nvo. In: Astronomical Data Analysis Software and Systems (ADASS) XIII (2003)
7. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.H., Vahi, K.: Characterization of scientific workflows. In: Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS '08) (2008)
8. Bruneman, P., Khanna, S., Tan, W.C.: Why and where: A characterization of data provenance. In: Proceedings of the 8th International Conference on Database Theory (2001)
9. Center, S.C.E.: Community modeling environment. http://www.scec.org/cme/
10. Chase, J.S., Irwin, D.E., Grit, L.E., Moore, J.D., Sprenkle, S.E.: Dynamic virtual clusters in a grid site manager. In: 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03) (2003)
11. Corral. http://pegasus.isi.edu/corral/latest
12. Dagman (directed acyclic graph manager). http://cs.wisc.edu/condor/dagman
13. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: An overview of workflow system features and capabilities. Future Generation Computer Systems **25(**5), 528–540 (2008)
14. Deelman, E., Livny, M., Mehta, G., Pavlo, A., Singh, G., Su, M.H., Vahi, K., Wenger, R.K.: Pegasus and DAGMan from Concept to Execution: Mapping Scientific Workflows onto Today's Cyberinfrastructure, pp. 56–74. IOS Press (2008)
15. Deelman, E., Singh, G., Livny, M., Berriman, B., Good, J.: The cost of doing science on the cloud: The montage example. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing (2008)
16. Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C., Katz, D.S.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Scientific Programming **13**(3), 219–237 (2005)
17. Foster, I., Freeman, T., Keahey, K., Scheftner, D., Sotomayer, B., Zhang, X.: Virtual clusters for grid communities. In: Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06) (2006)
18. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications **15**(3), 200–222 (2001)
19. Frey, J., Tannenbaum, T., Foster, I., Livny, M., Tuecke, S.: Condor-g: A computation management agent for multi-institutional grids. In: 10th International Symposium on High Performance Distributed Computing (2001)
20. Gentzsch, W.: Sun grid engine: Towards creating a compute power grid. In: Proceedings of the 1st International Symposium on Cluster Computing and the Grid (2001)
21. Gilbert, L., Tseng, J., Newman, R., Iqbal, S., Pepper, R., Celebioglu, O., Hsieh, J., Cobban, M.: Performance implications of virtualization and hyper-threading on high energy physics applications in a grid environment. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) (2005)

22. Glidein. http://www.cs.wisc.edu/condor/glidein
23. Groth, P., Deelman, E., Juve, G., Mehta, G., Berriman, B.: Pipeline-centric provenance model. In: Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science (WORKS '09) (2009)
24. Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J.: On the use of cloud computing for scientific workflows. In: Proceedings of the 3rd International Workshop on Scientific Workflows and Business Workflow Standards in e-Science (SWBES '08) (2008)
25. Inc., G.: Glusterfs. http://www.gluster.org
26. Inc, H.: Cloudstatus. http://www.cloudstatus.com
27. Inc, P.: Panasas. http://www.panasas.com
28. Juve, G., Deelman, E.: Resource provisioning options for large-scale scientific workflows. In: Proceedings of the 3rd International Workshop on Scientific Workflows and Business Workflow Standards in e-Science (SWBES '08) (2008)
29. Juve, G., Deelman, E., Vahi, K., Mehta, G.: Experiences with resource provisioning for scientific workflows using corral. Scientific Programming (to appear)
30. Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B.P., Maechling, P.: Scientific workflow applications on amazon ec2. In: Workshop on Cloud-based Services and Applications in conjunction with 5th IEEE International Conference on e-Science (e-Science '09) (2009)
31. Keahey, K., Freeman, T.: Contextualization: Providing one-click virtual clusters. In: Proceedings of the 4th International Conference on eScience (eScience '08) (2008)
32. Kee, Y., Kesselman, C., Nurmi, D., Wolski, R.: Enabling personal clusters on demand for batch resources using commodity software. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS '08) (2008)
33. Li, H., Ruan, J., Durbin, R.: Mapping short dna sequencing reads and calling variants using mapping quality scores. Genome Research **18**(11), 1851–1858 (2008)
34. Ligon, W.B., Ross, R.B.: Implementation and performance of a parallel file system for high performance distributed applications. In: Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing (1996)
35. Litzkow, M., Livny, M., Mutka, M.: Condor - a hunter of idle workstations. In: Proceedings of the 8th International Conference on Distributed Computing Systems (1988)
36. Microsystems, S.: Lustre. http://www.lustre.org
37. National center for supercomputing applications (ncsa). http://www.ncsa.illinois.edu
38. Open science grid. http://www.opensciencegrid.org
39. Palankar, M.R., Iamnitchi, A., Ripeanu, M., Garfinkel, S.: Amazon s3 for science grids: a viable solution? In: International workshop on Data-aware distributed computing (2008)
40. Pegasus workflow management system. http://pegasus.isi.edu
41. Raicu, I., Zhao, Y., Dumitrescu, C., Foster, I., Wilde, M.: Falkon: a fast and light-weight task execution framework. In: Proceedings of the 2007 ACM/IEEE conference on Supercomputing (2007)
42. Sapuntzakis, C., Brumley, D., Chandra, R., Zeldovich, N., Chow, J., Lam, M., Rosenblum, M.: Virtual appliances for deploying and maintaining software. In: Proceedings of the 17th USENIX conference on System administration (2003)
43. Schmuck, F., Haskin, R.: Gpfs: A shared-disk file system for large computing clusters. In: Proceedings of the 1st USENIX Conference on File and Storage Technologies (2002)
44. San diego supercomputing center (sdsc). http://www.sdsc.edu
45. Singh, G., Kesselman, C., Deelman, E.: Performance impact of resource provisioning on workflows. Tech. rep., University of Southern California, Information Sciences Institute (2005)
46. Singh, G., Kesselman, C., Deelman, E.: A provisioning model and its comparison with best-effort for performance-cost optimization in grids. In: Proceedings of the 16th international symposium on High performance distributed computing (HPDC '07) (2007)

47. Sotomayor, B., Childers, L.: Globus toolkit 4 programming Java services. Elsevier; Morgan Kaufmann (2006)
48. Teragrid. http://www.teragrid.org/
49. Youseff, L., Seymour, K., You, H., Dongarra, J., Wolski, R.: The impact of paravirtualized memory hierarchy on linear algebra computational kernels and software. In: Proceedings of the 17th international symposium on High performance distributed computing (2008)
50. Yu, W., Vetter, J.S.: Xen-based hpc: A parallel i/o perspective. In: Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '08) (2008)

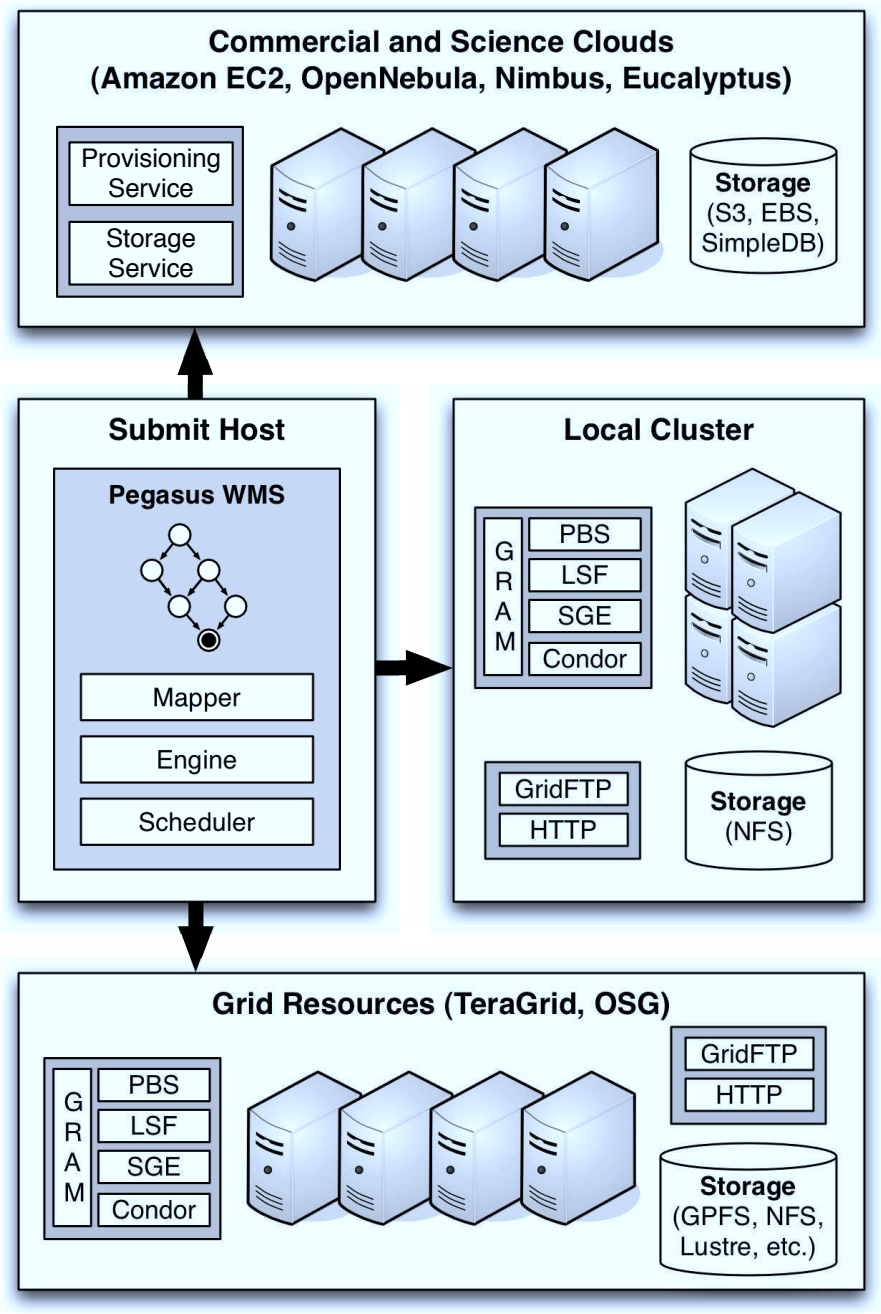


Fig. 1: The Workflow Management in the Context of the Execution Environment.

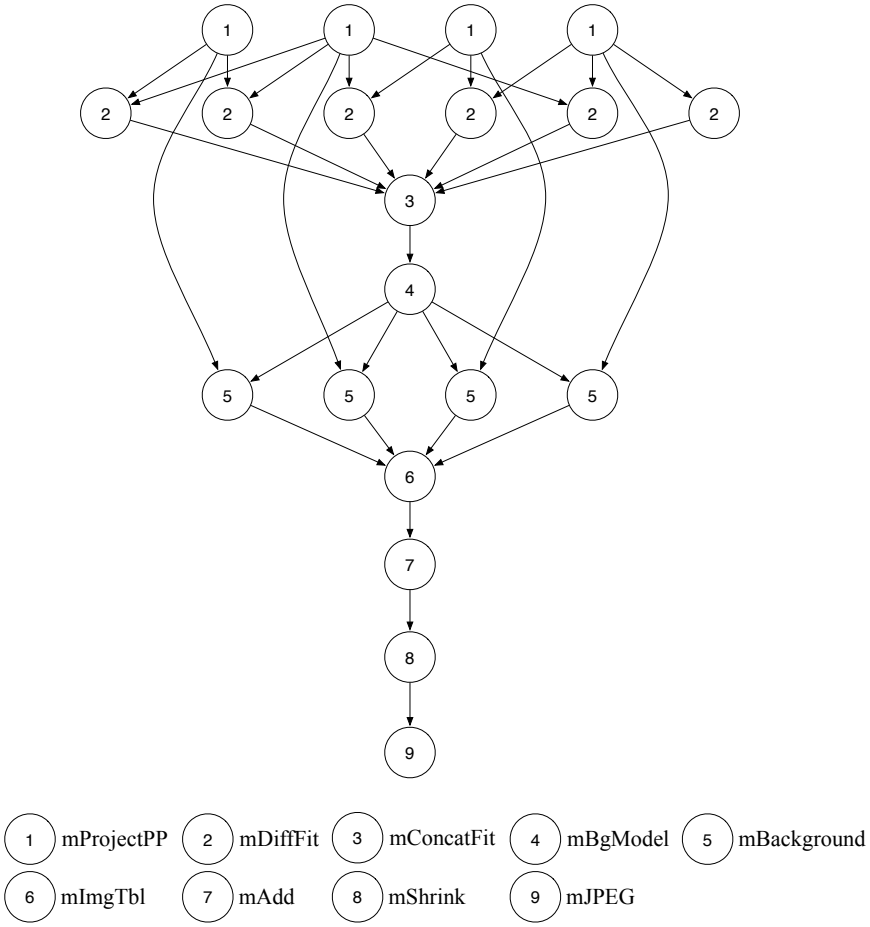


Fig. 2: Montage workflow.

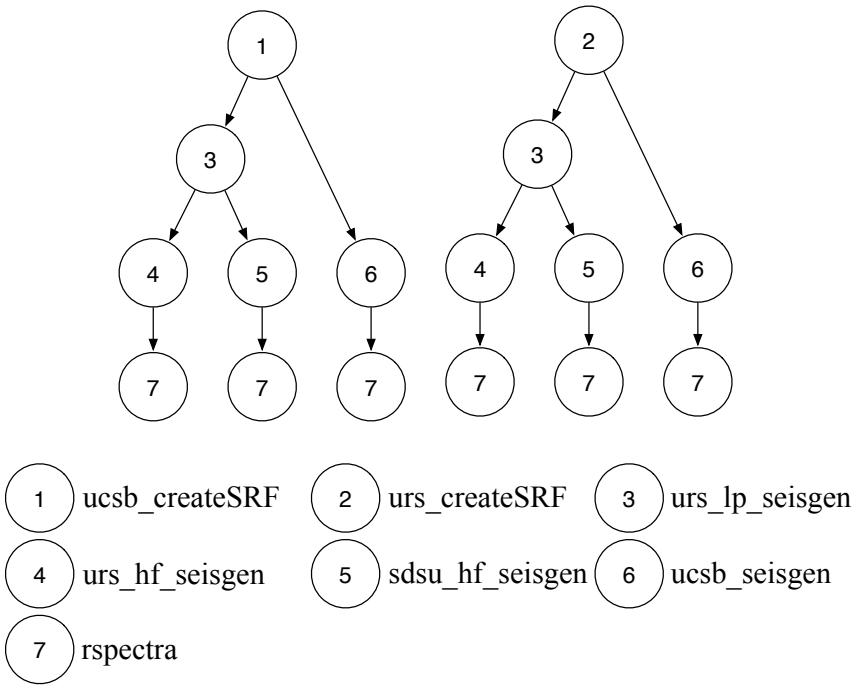


Fig. 3: Broadband Workflow.

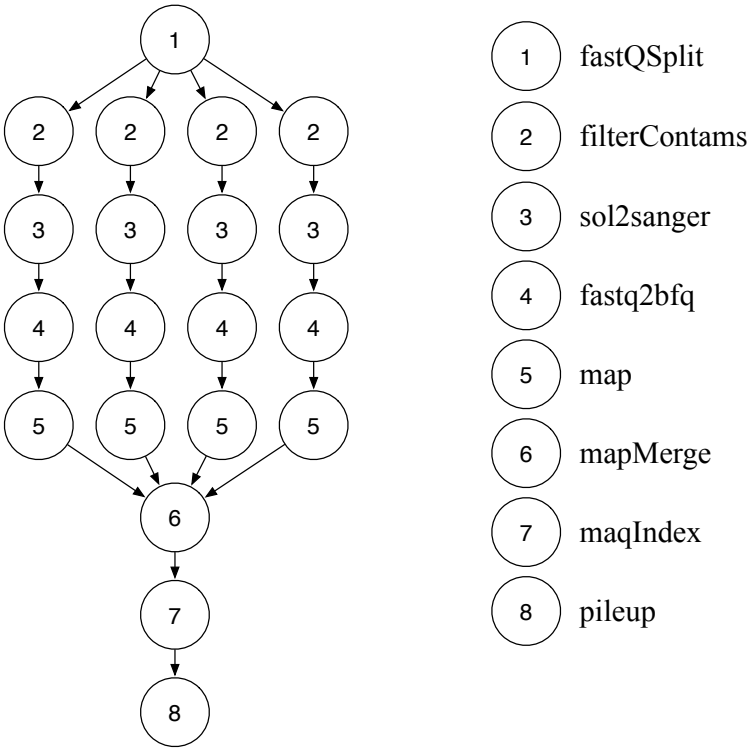
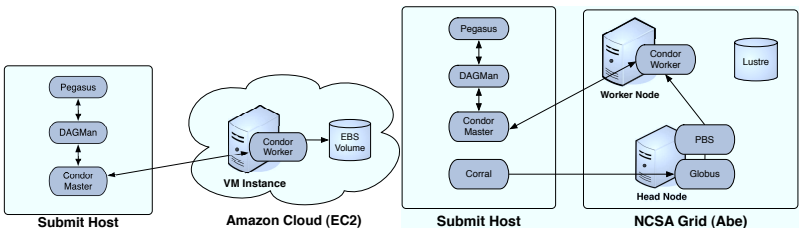


Fig. 4: Epigenome Workflow.



(a) on EC2 (b) on Abe

Fig. 5: Execution Environment

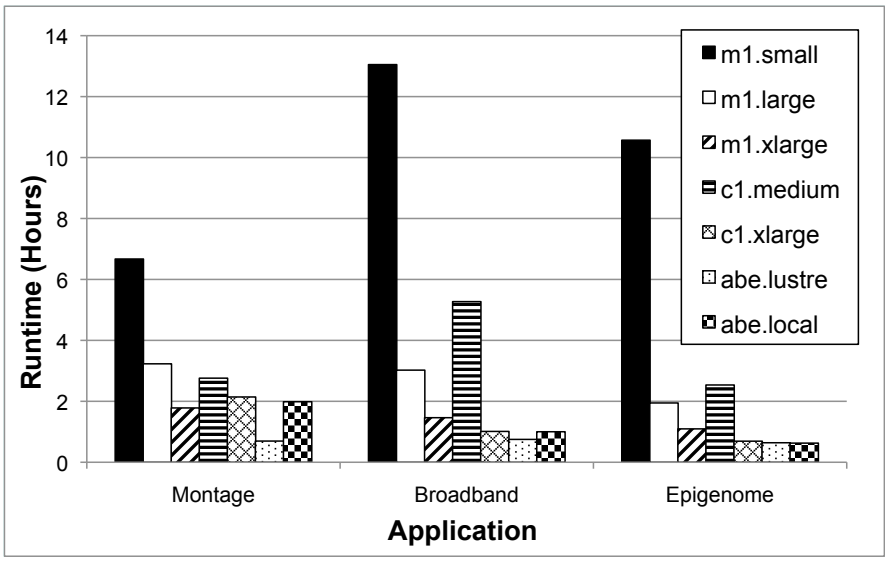


Fig. 6: Runtime Comparison.

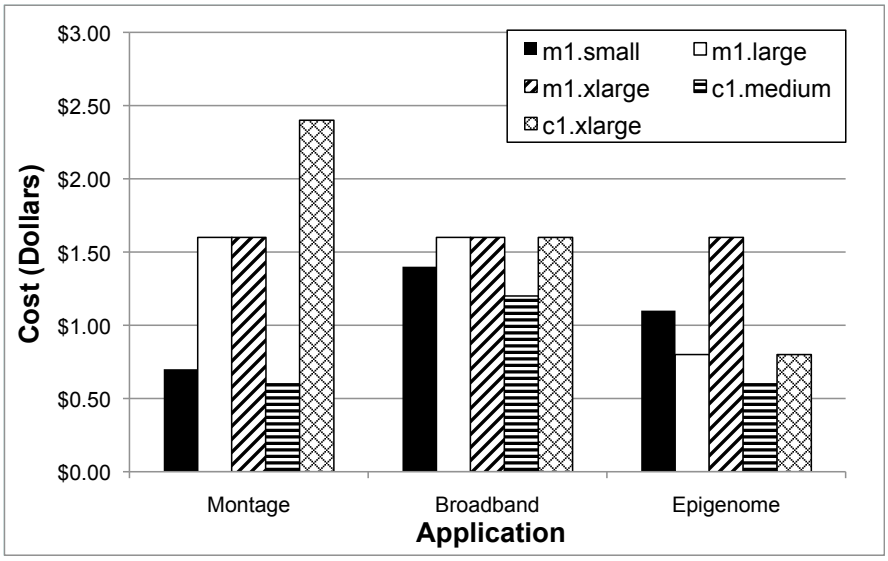


Fig. 7: Resource Cost Comparison.

Auspice: Automatic Service Planning in Cloud/Grid Environments

David Chiu and Gagan Agrawal

Abstract Recent scientific advances have fostered a mounting number of services and data sets available for utilization. These resources, though scattered across disparate locations, are often loosely-coupled both semantically and operationally. This loosely-coupled relationship implies the possibility of linking together operations and data sets to answer queries. This task, generally known as automatic service composition, therefore abstracts the process of complex scientific workflow planning from the user. We have been exploring a metadata-driven approach toward automatic service workflow composition, among other enabling mechanisms, in our system, Auspice: Automatic Service Planning in Cloud/Grid Environments. In this paper, we present a complete overview of our system's unique features and outlooks for future deployment as the Cloud computing paradigm becomes increasingly eminent in enabling scientific computing.

1 Introduction

Steady progress in various scientific fields including, but not limited to, geoinformatics [30, 29, 22], astronomy [41], bioinformatics [26, 27, 28], and high-energy physics [5], has led to a cornucopia of new data. These scientific data sets are typically stored in structured, low-level files for a variety of reasons despite the ongoing success in database technologies.[1] To store these vast collections of files, Mass Storage Systems (MSS) are typically employed. Reliant on voluminous, but slow, disk/-tape storage, MSS systems are distributed over existing networks including clusters, scientific Data Grids, and more recently, the Cloud [44], in combination that culmi-

Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210, USA

[1] For instance, a series of scientific observations is easily represented by arrays but not relational tables.

nates a "scientific Web." Figure 1 exemplifies this distribution model of scientific data storage and services.

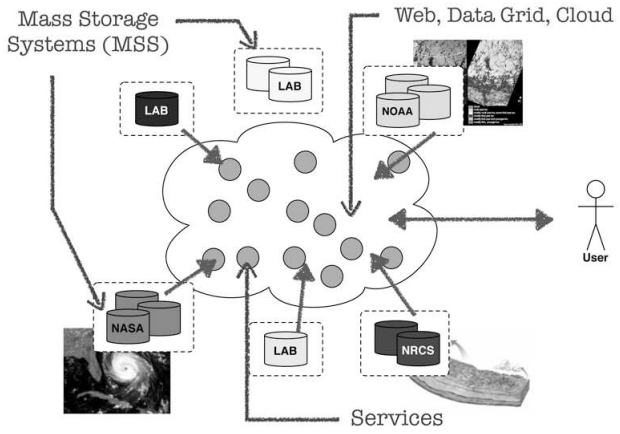


Fig. 1: Scientific Data Repositories and Services

At the same time, XML [45] emerged, a declarative markup language that allows common users to describe data sets. Having a accessible means for anyone to invent and provide data descriptions, metadata were eventually substantiated and standardized across many domains. As a result, metadata have become essential to many of today's applications. Because scientific data is often cryptic, the data management and Web communities have pressed for the coupling of metadata with certain data sets. The Dublin Core Metadata Initiative, for instance, has instituted a general set of cross-domain metadata elements (e.g., author, title, date, etc.) [18]. Attuned the importance of metadata, the scientific community also began undertaking tremendous efforts toward standardizing metadata formats. These efforts produced such formats as the Content Standard for Digital Geospatial Metadata (CSGDM) [21] and the Biological Data Profile (BDP) [39]. With a mounting number of reliable scientific metadata, relevant data sets can be identified and accessed more efficiently in today's cyberinfrastructures.

Meanwhile, the success of Service Oriented Architectures (SOA) has ushered an abundant deployment of interoperable processes/services, access to high-level utilities, and other compute and storage resources across scientific and geographic domains. These concrete advancements were instrumental in resurfacing the metacomputing framework, now known as the Grid [23], which is aptly named to reflect the vision of ubiquitous access to pervasive compute resources.

With the Grid's high availability of distributed data sets and services comes the nontrivial challenge for scientists and other end-users to manage such information. For instance, certain information involves execution of several operations, with disparate inputs, in a particular sequence. This process is typically known as service

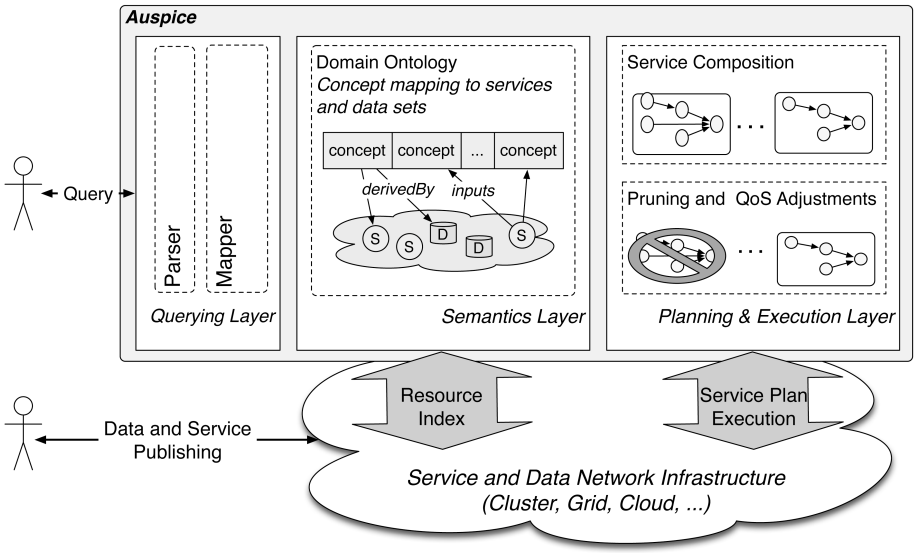


Fig. 2: Auspice System Architecture

composition. The Grid computing environment, together with the need to manage and process scientific data, triggered a resurgence of interest in workflows, which were originally employed for managing complex business operations. Whereas today's mainstream scientific workflow systems (e.g., Pegasus [19], Kepler [2], Triana [38], and Taverna [40], and others) have been instrumental in reducing the need for domain scientists to be familiar with the nuances of large-scale computing. For instance, most scientific workflow management systems provide high-level user interfaces for planning dependent operations, whereas computing provisions such as scheduling and resource allocation are hidden. Certainly, a goal for enabling these service level workflows is to automate their composition while simultaneously hiding such esoteric details as service and data discovery, integration, and scheduling from the common user.

1.1 Our Vision with Auspice

In this chapter, we discuss our contributions with Auspice (Automatic Service Planning in Cloud/Grid Environments), a metadata-driven service composition system developed at the Ohio State University. Auspice processes queries through automatic composition and execution of service workflow plans. Auspice is data-driven in the way that it leverages domain specific metadata to automatically derive loosely-coupled service plans that are semantically sound. Our system also enables Quality of Service through adaptive service planning, allowing it to scale execu-

tion times and data derivation accuracies to user needs. Auspice also exploits the emergence of the pay-as-you-go supercomputing infrastructure provisioned via the Cloud, by offering a flexible intermediate data cache. Our system, shown in Figure 2, comprises several independent layers, each encapsulated with certain functionalities. We outline the system in its entirety and, in the same effort, our research contributions.

The rest of this chapter is organized as follows. We first describe the Auspice metadata framework (Section 2) because it is central to our framework. Methods for planning, QoS handling, and caching are described in Section 3. Our approach for keyword querying integration is discussed in Section 4. A system evaluation was performed to show the effectiveness of our QoS handling and caching, two of Auspice's distinguishing features (Section 5). A discussion of related works is given in Section 6, and finally, we conclude in Section 7.

2 Metadata Framework

Auspice's Semantics Layer in Figure 2 facilitates the metadata functionalities necessary for our automatic service composition algorithm.

2.1 Capturing Concept Derivation

In designing the framework, our observation is that many scientific fields typically contain a set of domain-specific concepts, e.g., in geoinformatics: coordinates, bathymetry (water depth), coast lines, etc. These domain concepts can typically be derived by the available data sets and services. Users, who have become increasingly goal-oriented, target these concepts in their queries: an engineer might prefer a system capable of immediately answering

```
''what is the coast line along coordinate x?''
```

rather than having to find and orchestrate the execution of a workflow composing several operations using files from several potentially disparate data sources. Abstractly, one can envision a "concept derivation" scheme as a means to automatically generate plans, composing the necessary service operations and data sets to answer queries. That is, a concept, c, is derived by a service s, whose inputs, (x,y,z) are again substantiated by concepts c_x, c_y, c_z, etc., until a terminal element (either a service without input or data file) has been reached.

We proposed an ontology to capture these concept derivation relationships: Let ontology $O = (V_O, E_O)$ be a directed acyclic graph where its set of vertices, V_O, comprises a disjoint set of classes: concepts C, services S, and data types, D, i.e., $V_O = (C \cup S \cup D)$. Each directed edge $(u,v) \in E_O$ must denote one of the following relations:

- $(u\ \delta^{c\to s}\ v)$: concept-service derivation. Service $u \in S$ used to derive $v \in C$.
- $(u\ \delta^{c\to d}\ v)$: concept-data type derivation. Data type $u \in D$ used to derive $v \in C$.
- $(u\ \delta^{s\to c}\ v)$: service-concept derivation. Concept $u \in C$ used to derive service $v \in S$.

Planning a workflow with this structure in place becomes possible as the task is simplified to depth-first search from the query's targeted concept node. In the top half of Figure 3, we show one possible subset of the ontology *vis-à-vis* with the coast line target concept.[2] Although the abstract workflow is not shown, it can be easily envisioned by reversing the arrows (depicting derivation dependency), and removing the intermediate concept nodes. The bottom half of Figure 3 shows the executable, concrete workflow after some substantiation of the required data (and query parameters).

2.2 Enabling Fast Resource Identification

Although the aforementioned semantics framework can be used to find a concept's overall derivation structure, inputs and files specific to the user's query has yet to be determined. This is by far the most time-consuming aspect of the planning algorithm. But to reduce its complexity, we apply a metadata indexing scheme for data sets and services. In a process we call *metadata registration*, each known file is indexed by their respective popular identifying concepts extracted from their corresponding metadata. For instance, a file in the geographical domain is typically identified by the time and location that it represents. The general metadata registration framework allows domain scientists to define rules for the concept's extraction from each metadata format, and builds a unified index to quickly identify the necessary files for workflow construction. Service metadata registration, although less crucial since $|D| \gg |S|$, is also employed to index their derivation and input needs.

On receiving the above coast line query, we have shown that Auspice can accurately plan the workflow toward its derivation (given a set of known services and data sets), as well as quickly identifying the necessary files associated with coordinate x. A nuanced discussion of this example within the semantics framework can also be found in [11].

3 Service Workflow Planning

In this section, we begin the discussion of the system's Planning and Execution Layer, responsible for automatic service composition, making QoS decisions, and workflow execution. The ontology described in the previous section is capable of

[2] Other derivation paths may exist within a certain ontology, but for simplicity, we show just one here.

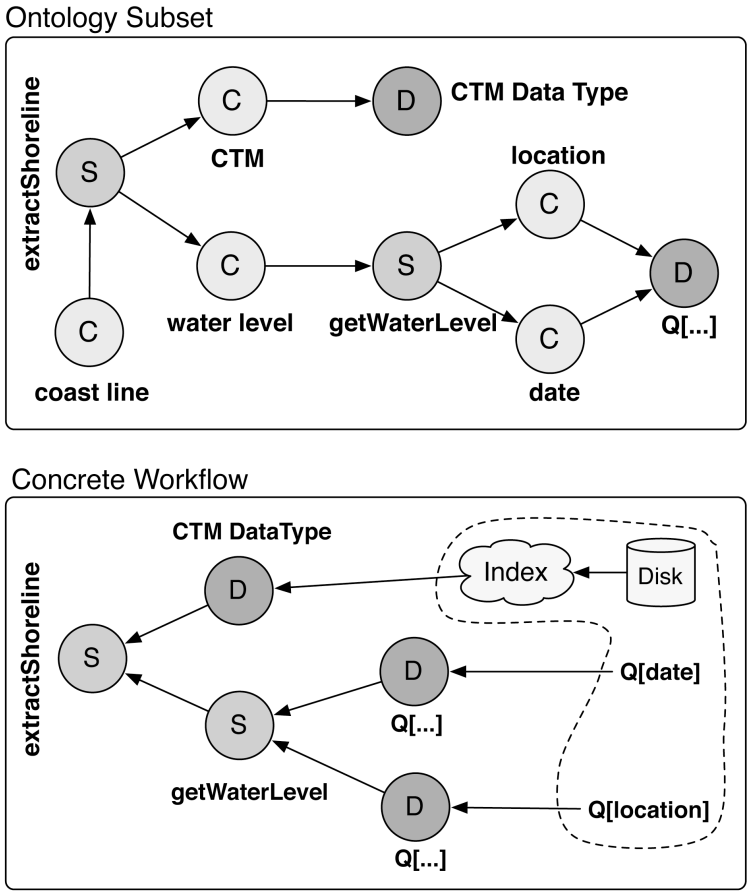


Fig. 3: Coast Line Workflow Example

defining workflows in the recursive form of composite derivations. In the same way, a workflow w can be defined in our system as follows:

$$w = \begin{cases} \varepsilon \\ d \in D \\ (op, P_{op}) \in S \end{cases} \tag{1}$$

such that terminals ε and $d \in D$ denote a null workflow and a data instance (file, user input, intermediate data, etc.) belonging to a specific data type in D, respectively. Nonterminal $(op, P_{op}) \in S$ is a tuple where op denotes a service operation with a corresponding parameter list $P_{op} = (p_1, \ldots, p_k)$ and each p_i is itself a workflow. In other words, a workflow is a tuple that either contains a single data instance or a service operation whose parameters are, recursively, (sub)workflows.

Our planning algorithm, called workflow enumeration (WFEnum), takes as input the targeted concept, t and $Q[\ldots]$, the mapped list of query parameters such that $Q[k] \rightarrow val \mid k \in C$. The planner's goal is to enumerate all possible service plans that can be used to somehow derive t with respect to the given Q. Algorithm 1 shows a condensed version of the WFEnum algorithm originally proposed in [11]. While the overall algorithm has been reduced here, its basic logic is still conveyed. Faithful to the workflow structure defined in Equation 1, WFEnum generates and returns a list of possible workflows, $W = (w_1, \ldots, w_m)$, by first deriving t via any data sets available (Lines 2-8). Finally, WFEnum searches all service-derived paths (Lines 9-18) by composing all services known to derive t. The composition of the services is driven by a recursive reduction of the concepts pertaining to all of its input parameters. Whereas the ontology guides the derivation links (Line 2 for data types and Line 9 for services), the metadata index, implicit in the algorithm, identifies the files efficiently (Line 4).

Algorithm 1 WFEnum(t, Q)

```
 1: W ← ()
 2: for all {(t,v) ∈ G | v ∈ D} do
 3:     //v = d ∈ D
 4:     F ← σ_<Q>(d) //select files w.r.t. Q
 5:     for all f ∈ F do
 6:         W ← W ∪ {f}
 7:     end for
 8: end for
 9: for all {(t,v) ∈ G | v ∈ S} do
10:     //v = (op, P_op) ∈ S
11:     W_op ← ()
12:     for all p ∈ P_op do
13:         W_op ← W_op × WFEnum(p.t, Q)
14:     end for
15:     for all pm ∈ W_op do
16:         W ← W ∪ {(op, pm)}
17:     end for
18: end for
19: return W
```

3.1 Planning with QoS Adaptivity

Often, there exists multiple ways to derive a given query, using different combinations of data sources and services. Which costs differentiate each workflow, then, should be determined. In most scientific domains, we are concerned with two Qualities of Service constraints: execution time and the accuracy of results.

In terms of time, it is expected that users may have certain constraints on execution time. But in heterogeneous networks, such as geographically diverse Grids, ex-

ecution times cannot always be guaranteed. In our earlier work [15], we showed that Auspice can adapt to heterogeneity in underlying networks with varying bandwidths by dynamically reducing the complexity of data sets in the workflow. Reducing data sets, however, introduced a vexing problem of finding the corresponding errors with respect to the scientific domain. We return to the running coast line query example. Let us assume that the user now requests that the coast line be returned in some T amount of time, with a mean error no greater than E meters in length. Now, consider a case where Auspice determined a workflow can be completed in T time by reducing the resolution of some image by α%. The problem becomes finding the relationship between the system accuracy parameter, α, and the domain-specific error, E, for example, difference in meters.

We studied this effect in [14, 16], and proposed a general approach to automatically assign system-based adaptive parameters, such as an α resolution rate, in order to still meet the user-based accuracy/error parameters, E. We found that many scientific error models are complex and implemented by read-only algorithms. Clearly, given some scientific error model, $\sigma(\alpha)$ that estimates E on varying resolution values it is not always possible to inverse σ for providing a precise α. We proposed that, given an error constraint, E, σ can be iteratively invoked on disparate values $(v_1, \ldots, v_i)$ until convergence, i.e., $E \approx \sigma(v_i)$ and trivially, $\alpha \approx v_i$. We showed, in [16], that v_i can be found via a binary-search approach, and α-convergence is on the order of microseconds, which is negligible when compared to the overall planning time.

3.2 Flexible Derived Data Caching

As scientific data sets continue to mount, the overall execution times of large-scale workflows clearly becomes dominated by computation and network transfer times. In an effort to reduce execution times, we believe that a traditional approach, caching, can be applied to the Auspice framework. In [13], we built a hierarchical cache for intermediate derived workflow data. This cache uses a modified version of B^x-Trees [33], to index already derived data. Our cache was implemented on a cluster, and was shown to not only scale well to an increasing number of cluster nodes, but also speed up our queries accordingly.

To bring this idea a step beyond clusters and into today's Cloud infrastructure [4], we surmise that technologies developed for Web caching can be employed to build a flexible hierarchical cache. A Cloud-based cache will be flexible in the sense that not only can it dynamically allocate Cloud nodes to grow in size, but it should also shrink to save cost for Cloud usage. We utilize consistent hashing [34], which has found application in Web proxies, among others. With all forms of hashing, the dynamic allocation (and deallocation) of nodes causes overhead due to the migration of indexes to new nodes. Consistent hashing was developed to reduce this overhead, and hence is friendly to dynamic underlying infrastructures such as the Cloud. For Cloud-use, we proposed an algorithm, Greedy Bucket Allocation (GBA), which

allocates Cloud nodes as a last resort to save costs while managing to cache as many intermediate data sets as possible [12]. We have shown that on-demand machine allocation for intermediate data caching within a simulated Cloud not only improves execution times, but also minimizes the potentials for underutilizing resources (not incidentally, it also helps reduce the Cloud utilization cost). While our algorithms were able to scale up resources when needed, the Cloud computing paradigm ushers in a new dimension in cost optimization. That is, applications should also scale down to save cost. However, this decision is difficult to make – data and job migration costs are high, so down-scaling should only be performed when it is predictable that potentials for increasing workloads are not imminent.

We implement a cache contraction scheme to merge nodes when query intensities are lowered. Our scheme is based on a combination of exponential decay and a temporal sliding window. Because the size of our cache system (number of nodes) is highly dependent on the frequency of queries during some timespan, we propose a global cache eviction scheme that captures querying behavior. In our contraction scheme, we employ a streaming model, where incoming query requests represent streaming data, and a global view of the most recently queried keys is maintained in a sliding window. A sliding window, $T = (t_1, \ldots, t_m)$, comprises m time slices of some fixed real-time length. Each time slice, t_i, associates a set of keys queried in the duration of that slice. We argue that, as time passes, older unreferenced keys (i.e., t_i nearing t_m) should have a lower probability of existing in the cache. As these less relevant keys become evicted, the system makes room for newer, incoming keys (i.e., t_i nearing t_1) and thus capturing temporal locality of the queries.

Cache eviction occurs when a time slice has reached t_{m+1}, and at this time, an eviction score,

$$\lambda(k) = \sum_{i=1}^{m} \alpha^{i-1} |\{k \in t_i\}|$$

is computed for every key, k, within the expired slice. The ratio, $\alpha : 0 < \alpha < 1$, is a *decay* factor, and $|\{k \in t_i\}|$ returns the number of times k appears in some slice t_i. Here, α is passive in the sense that a higher value corresponds to a larger amount of keys that is kept in the system. After λ has been computed for each key in t_{m+1}, any key whose λ falls below the threshold, T_λ, is evicted from the system. Notice that α is amortized in the older time slices, in other words, recent queries for k are rewarded, so k is less likely to be evicted. Clearly, the sliding window eviction method is sensitive to the values of α and m. A baseline value for T_λ would be α^{m-1}, which will not allow the system to evict any key if it was queried even just once in the span of the sliding window. We will show their effects in the experimental section.

Due to the eviction strategy, a set of cache nodes may eventually become lightly loaded, which is an opportunity to scale our system down. The nodes' indices can be merged, and subsequently, the superfluous node instances can be discarded.

4 Keyword Querying

The Querying Layer in Auspice is responsible for decomposing a user's query into concepts within the ontology, as well as materializing the concepts with the user's given values. Keyword search, without saying, has become a mainstay for common querying. Albeit that abundant effort has been put into supporting keyword searches in general unstructured documents, e.g., the Web, the current technologies are quite excessive in the context of our system. Auspice encompasses domain-specific information, a quality generally missing from the Web. Auspice's knowledge framework, including metadata and the ontology (discussed in the Semantics Layer), can be employed here to better facilitate keyword queries. We believe that this is the first keyword-search endeavor into automatic service composition and scientific workflow systems.

4.1 Keyword-Maximization Query Planning

To support keyword queries, we automatically compose all workflows relevant to the most number of keywords in the user query, K. We currently support only AND-style keyword queries. Auspice's querying algorithm returns all workflow plans, w, whose concept-derivation graph, $\psi(w)$ (to be discussed later), contains the most concepts from K, while under the constraints of the user's query parameters, Q. To exemplify the algorithms, we prescribe the ontology subset shown in Figure 4 to our discussion. Furthermore, we interweave the description of the algorithms with the keyword query example:

```
''wind coast line CTM image (41.48335,-82.687778) 8/20/2009''
```

Here, we note that the given coordinates point to Sandusky, Ohio, a location where we have abundant data sets.

The data and service metadata registration procedure, discussed previously, allows the user to supply some keywords that describe their data set or the output of the service. These supplied keywords are used to identify the concepts in which the new resource derives, and if such a concept does not exist, the user is given an option to create one in the ontology. As such, each concept, c, has an associated set of keywords, K_c. For instance, the concept of *elevation* might associate $K_{elevation} = \{$ "height", "elevation", "DEM"$\}$. The WordNet database [20] was also employed to expand K_c for the inclusion of each term's synonyms.

Some terms, however, can only be matched by their patterns. For example, "13:00" should be mapped to the concept, *time*. Others require further processing. A coordinate, (x,y), is first parsed and assigned concepts independently, (i.e., $x \leftarrow$ longitude and $y \leftarrow$ latitude). Because Auspice is currently implemented over the geospatial domain, only a limited number of patterns are expected. Finally, the last pattern involves value assignment. In our keyword system, values can be given directly to concepts using a *keyword=value* string. That is, the keyword query, "water

level (x,y)" is equivalent to "water level latitude=y longitude=x". Finally, each query term is matched against this set of terms to identify their corresponding concepts. Indeed, a keyword may correspond with more than one concept. However, to be discussed next, using the concept-derivation of the keyword concepts, our algorithm prunes all unlikely terms during the workflow planning phase.

Before we describe the workflow enumeration algorithm, WFEnum_key (shown as Algorithm 2), we introduce the notion of concept derivation graphs (or ψ-graphs) which is instrumental in WFEnum_key for pruning. ψ-graphs are obtained as concept-derivation relationships, $\psi(c) = (V_\psi, E_\psi)$, where c is a concept, from the ontology. All vertices within $\psi(c)$ denote only concepts, and its edges represent derivation paths. As an aside, ψ can also be applied on workflows, i.e., $\psi(w)$ extracts the concept-derivation paths from the services and data sets involved in w.

4.2 Planning Algorithm

WFEnum_key's inputs include c_t, which denotes the targeted concept. That is, all generated workflows, w, must have a ψ-graph rooted in concept c_t. Specifically, only workflows, w, whose $\psi(w) \subseteq \psi(c_t)$ will be considered for the result set. The next input, Φ, is a set of required concepts, and every concept in Φ must be included in the derivation graph of c_t. A set of query parameters, Q, is also given to this algorithm. These would include the coordinates and the date given by the user in our example query. Q is used to identify the correct files and also as input into services that require these particular concept values. Finally, the ontology, O, supplies the algorithm with the derivation graph.

On Lines 2-8, the planning algorithm first considers all data-type derivation possibilities within the ontology for c_t, e.g., $(c_t\ \delta^{c\to d}\ d_t)$. All data files are retrieved with respect to data type d_t and the parameters given in Q. Each returned file record, f, is an independent file-based workflow deriving t. Next, the algorithm handles service-based derivations. From the ontology, O, all $(c_t\ \delta^{c\to s}\ s_t)$ relations are retrieved. Then for each service, s_t, that derives c_t, its parameters must first be recursively planned. Line 15 thus retrieves all concept derivation edges $(s_t\ \delta^{s\to c}\ c_{s_t})$ for each of its parameters. Opportunities for pruning are abundant here. For instance, if the required set of concepts, Φ, is not included in the ψ-graphs of all s_t's parameters combined, then s_t can be pruned because it does not meet the query's requirements. For example, on the bottom left corner of Figure 4, we can imply that another service, *img2*, also derives the *image* concept. Assuming that $\Phi = \{shore\}$, because the ψ-graphs pertaining to all of *img2*'s parameters does not account for the elements in Φ, *img2* can be immediately pruned here (Line 17). Otherwise, service s_t is deemed promising, and its parameters' concepts are used as targets to generate workflow (sub)plans toward the total realization of s_t. Recalling the workflow's recursive definition from previously, this step is tantamount to deriving the nonterminal case where $(s_t, (w_1, \ldots, w_p)) \in S$. Finally whereas the complete plan for s_t is included in the result set (Line 27), W, each (sub)plan is also included because they

Algorithm 2 WFEnum_key(c_t, Φ, Q, O)

```
1: static W
2: for all concept-data derivation edges w.r.t. c_t, (c_t δ^{c→d} d_t) ∈ E_O do
3:     # data type d_t derives c_t; build on d_t
4:     F ← σ_{<Q>}(d_t) //select files w.r.t. Q
5:     for all f ∈ F do
6:         W ← W ∪ {f}
7:     end for
8: end for
9: # any workflow enumerated must be reachable within Φ
10: for all concept-service derivation edges w.r.t. c_t, (c_t δ^{c→s} s_t) ∈ E_O do
11:     # service s_t derives c_t; build on s_t
12:     W_{s_t} ← ()
13:     # remove target, c_t, from requirement set (since we current see it)
14:     Φ ← {Φ \ c_t}
15:     for all service-concept derivation edges w.r.t. s_t, (s_t δ^{s→c} c_{s_t}) ∈ E_O do
16:         # prune if elements in Φ do not exist in c_{s_t}'s derivation path, that is, the union of all its
          parents' ψ graphs
17:         if (Φ ⊆ ⋃ψ(c_{s_t})) then
18:             W' ← WFEnum_key(c_{s_t}, Φ ∩ ψ(c_{s_t}), Q, W, O)
19:             if W' ≠ () then
20:                 W_{s_t} ← W_{s_t} × W'
21:                 W ← W ∪ W'
22:             end if
23:         end if
24:     end for
25:     # construct service invocation plan for each p ∈ W_{s_t}, and append to W
26:     for all p ∈ W_{s_t} do
27:         W ← W ∪ {(s_t, p)}
28:     end for
29: end for
30: return W
```

include some subset of Φ, the required keyword concepts and therefore could be somewhat relevant to the user's query (Line 21).

With the planning algorithm in place, the natural extension now is to determine its input from a given list of keywords.

The query planning algorithm, shown in Algorithm 3, simply takes a set of keywords, K, and the ontology, O, as input, and the resulting list of workflow plans, R, is returned. First, the set of query parameters, Q_K, is identified using the concept pattern mapper on each of the key terms. Because user-issued parameter values are essentially data, they define a $\delta^{c\to d}$-type derivation on the concepts to which they are mapped. Here, $(longitude\ \delta^{c\to d}\ x)$, $(latitude\ \delta^{c\to d}\ y)$, $(date\ \delta^{c\to d}\ 8/20/2009)$, can be identified as a result (Line 2). The remaining concepts from K are also determined, $C_K = \{wind, shore, image, coastal\text{-}terrain\text{-}model\}$ (note that "coast" had been deduced to the concept, *shore*, and that "line" had been dropped since it did not match any concepts in O).

Next (Lines 5-14), the algorithm attempts to plan workflows incorporating all possible combinations of concepts within C_K. The power set, $\mathscr{P}(C_K)$ is computed

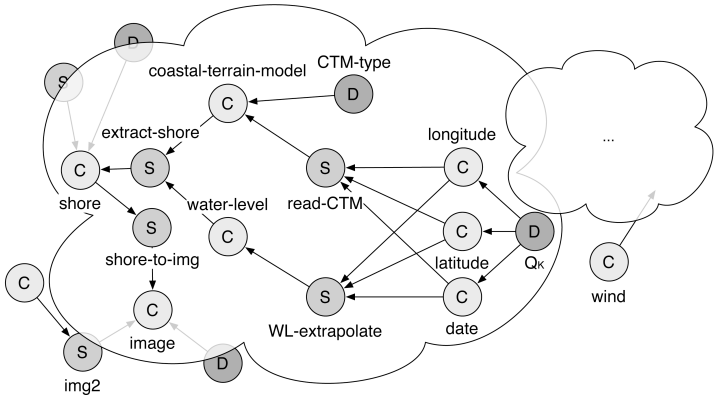


Fig. 4: An Exemplifying Ontology

Algorithm 3 KMQuery(K, O)

```
1:  R ← ()        # R will hold the list of derived workflow results
2:  Q_K ← O.mapParams(K)
3:  C_K ← O.mapConcepts(K \ Q_K)
4:  # compute the power set, 𝒫(C_K), of C_K
5:  for all ρ ∈ 𝒫(C_K), in descending order of |ρ| do
6:      # ρ = {c_1,...,c_n},{c_1,...,c_{n-1}},...,{c_1}
7:      # check for reachability within ρ, and find successor if true
8:      reachable ← false
9:      for all c_i ∈ ρ ∧ ¬reachable do
10:         if (ρ \ {c_i}) ⊆ ψ(c_i) then
11:             c_root ← c_i
12:             reachable ← true
13:         end if
14:     end for
15:     if reachable then
16:         # from ontology, enumerate all plans with c_root as target
17:         R ← R ∪ WFEnum_key(c_root, (ρ \ {c_root}), Q_K, O)
18:         # prune all subsumed elements from 𝒫(C_K)
19:         for all ρ' ∈ 𝒫(C_K) do
20:             if ρ' ⊆ ρ then
21:                 𝒫(C_K) ← 𝒫(C_K) \ {ρ'}
22:             end if
23:         end for
24:     end if
25: end for
26: return R
```

for C_K, to contain the set of all subsets of C_K. Then, for each subset-element $\rho \in \mathscr{P}(C_K)$, the algorithm attempts to find the root concept in the derivation graph produced by ρ. For example, when $\rho = \{shore, image, coastal\text{-}terrain\text{-}model\}$, the root concept is *image* in Figure 4. However, when $\rho = \{shore, coastal\text{-}terrain\text{-}model\}$, then $c_{root} = shore$. But since any workflows produced by the former subsumes any produced by the latter ρ set of concepts, the latter can be pruned (thus why we loop from descending order of $|\rho|$ on Line 5). In order to perform the root-concept test, for each concept element, $c_i \in \rho$, its ψ-graph, $\psi(c_i)$ is first computed, and if it consumes all other concepts in ρ, then c_i is determined to be the root (recall that $\psi(c_i)$ generates a concept-derivation DAG rooted in c_i).

Back to our example, although *wind* is a valid concept in O, it does not contribute to the derivation of any of the relevant elements. Therefore, when $\rho = \{wind, image, shore, coastal\text{-}terrain\text{-}model\}$, no plans will be produced because *wind* is never reachable regardless of which concepts is considered root. The next ρ, however, produces $\{image, shore, coastal\text{-}terrain\text{-}model\}$. Here, $\psi(image)$ incorporates both *shore* and *coastal-terrain-model*, and thus, *image* is determined to be c_{root}. The inner loop on Line 9 can stop here, because the DAG properties of O does not permit $\psi(shore)$ or ψ(coastal-terrain-model) to include *shore*, and therefore neither can be root for this particular ρ.

When a reachable ρ subset has been determined, the planning method, WFEnum_key can be invoked (Lines 15-24). Using c_{root} as the targeted with $\rho \setminus \{c_{root}\}$ being the concepts required in the derivation paths toward c_{root}, WFEnum_key is employed to return all workflow plans. But as we saw in Algorithm 1, WFEnum_key also returns any workflow (sub)plans that were used to derive the target. That is, although *image* is the target here, the *shore* concept would have to be first derived to substantiate it, and it would thus be included in R as a separate plan. Due to this redundancy, after WFEnum_key has been invoked, Lines 18-23 prunes the redundant ρ's from the power set. In our example, every subset element will be pruned except when $\rho = \{wind\}$. Therefore, *wind* would become rooted its workflows will likewise be planned separately.

4.3 Relevance Ranking

The resulting workflow plans should be ordered by their relevance. Relevance, however, is a somewhat loose term under our context. We define relevance as a function of the number of keyword-concepts that appear in each workflow plan. We, for instance, would expect that any workflow rooted in *wind* be less relevant to the user than the plans which include significantly more keyword-concepts: *shore*, *image*, etc. Given a workflow plan, w, and query, K, we measure w's relevance score, as follows:

$$r(w,K) = \frac{|V_\psi(w) \cap C(K)|}{|C(K)| + \log(|C(K) \setminus V_\psi(w)| + 1)}$$

Recall that $V_\psi(w)$ denotes the set of concept vertices in w's concept derivation graph, $\psi(w)$. Here, $C(K)$ represents the set of concept nodes mapped from K. This equation corresponds to the ratio of the amount of concepts from $C(K)$ that w captures. The log term in the denominator signifies a slight *fuzziness* penalty for each concept in w's derivation graph that was not specified in K. The motivation for this penalty is to reward "tighter" workflow plans are that more neatly represented (and thus, more easily understandable and interpreted by the user). This metric is inspired by traditional approaches for answering keyword queries over relational databases [43, 1].

4.4 A Case Study

We present a case study of our keyword search functionality in this section. Our system is run on an Ubuntu Linux machine with a Pentium 4 3.00Ghz Dual Core with 1GB of RAM. This work has been a cooperative effort with the Department of Civil and Environmental Engineering and Geodetic Sciences here at the Ohio State University. Our collaborators supplied us with various services that they had developed to process certain types of geospatial data. A set of geospatial data was also given to us. In all, the ontology used in this experiment consists of 29 concepts, 25 services, 5 data types. The 25 services and 2248 data files were registered to the ontology based on their accompanying metadata, solely for the purposes of this experiment. We note that, although the resource size is small, the given is sufficient for evaluating the *functionality* of keyword search support. A set of queries, shown in Table 1, are used to evaluate our system.

Table 1: Experimental Queries

Query ID	Description
1	"coast line CTM 7/8/2003 (41.48335,-82.687778)"
2	"bluff line DEM 7/8/2003 (41.48335,-82.687778)"
3	"(41.48335,-82.687778) 7/8/2003 wind CTM"
4	"waterlevel=174.7cm water surface 7/8/2003 (41.48335,-82.687778)"
5	"waterlevel (41.48335,-82.687778) 13:00:00 3/3/2009"
6	"land surface change (41.48335,-82.687778) 7/8/2003 7/7/2004"

First, we present the search time of the six queries issued to the system. In this experiment, we executed the search using two versions of our algorithm. Here, the search time is the sum of the runtimes for KMQuery and WFEnum_key algorithms. The first version consists of the a-priori pruning logic, and the second version does not prune until the very end. The results of this experiment are shown in Figure 5, and as we can see, a typical search executes on the order of several milliseconds, albeit that the ontology size is quite small.

We can also see that the pruning version results in slightly faster search times in almost all queries, with the exception of QueryID=3. It was later verified that this query does not benefit from pruning with the given services and data sets. In other words, the pruning logic is an overhead for this case. Along the right *y*-axis, the result set size is shown. Because the test data set is given by our collaborators, in addition to the fact that our search algorithm is exhaustive, we can claim (and it was later verified) that the recall is 100%. Recall by itself, however, is not sufficient to measuring the effectiveness of the search.

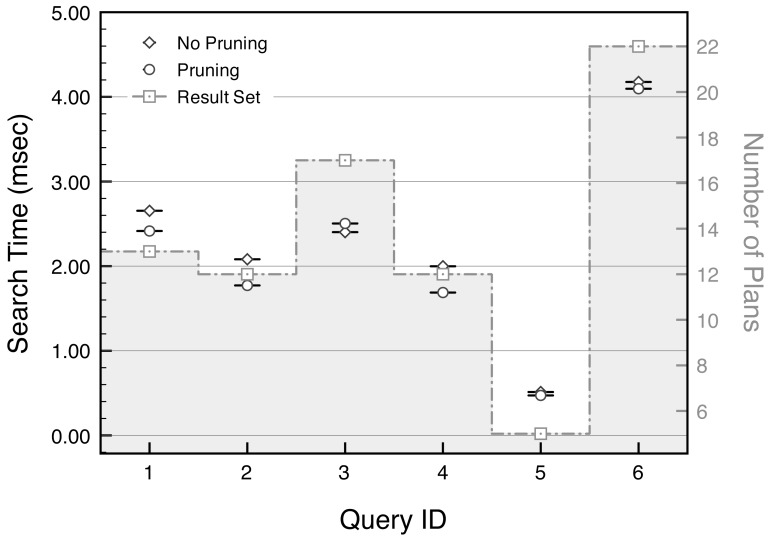


Fig. 5: Search Time

To measure the precision of the result set, we again required the help of our collaborators. For each workflow plan, w in the result set, the domain experts assigned a score, $r'(w,K)$ from 0 to 1. The precision for each plan is then measured relative to the difference of this score to the relevance score, $r(w,K)$, assigned by our search engine. For a result set R, its precision is thus computed,

$$prec(R,K) = \frac{1}{|R|} \sum_{w \in R} 1 - (|r(w,K) - r'(w,K)|)$$

The precision for our queries is plotted in Figure 6. Most of the variance are introduced due to the fact that our system underestimated the relevance of some plans. Because Query 3 appeared to have performed the worst, we show its results in Table 2. The third query contains five concepts after keyword-concept mapping: *wind*, *date*, *longitude*, *latitude*, and *coastal-terrain-model*. The first five plans enumerated captures all five concepts plus “water surface”, which is superfluous to the keyword

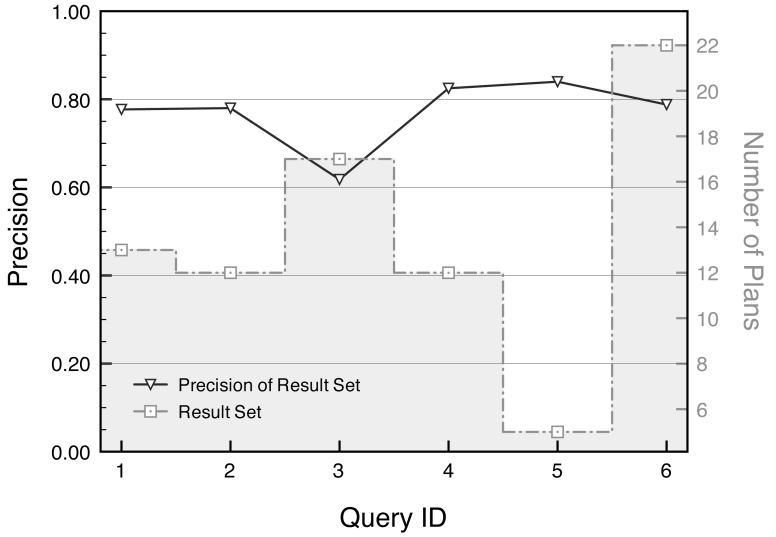


Fig. 6: Precision of Search Results

Table 2: QueryID 3 Results Set and Precision

Workflow Plan	r	r'
GetWindVal(GetWaterSurface(getCTMLowRes(CTM42.dat)))	0.943	0.8
GetWindVal(GetWaterSurface(getCTMMedRes(CTM42.dat)))	0.943	0.8
GetWindVal(GetWaterSurface(getCTMHighRes(CTM42.dat)))	0.943	0.8
GetWindVal(GetWaterSurface(CreateFromUrlLowRes(CTM42.dat)))	0.943	0.8
GetWindVal(GetWaterSurface(CreateFromUrlHighRes(CTM42.dat)))	0.943	0.8
getCTMLowRes(CTM42.dat)	0.8	0.3
getCTMMedRes(CTM42.dat)	0.8	0.3
getCTMHighRes(CTM42.dat)	0.8	0.3
CreateFromUrlLowRes(CTM42.dat)	0.8	0.3
CreateFromUrlHighRes(CTM42.dat)	0.8	0.3
CTM42.dat	0.8	0.3
GetWaterSurface(getCTMLowRes(CTM42.dat))	0.755	0.3
GetWaterSurface(getCTMMedRes(CTM42.dat))	0.755	0.3
GetWaterSurface(getCTMHighRes(CTM42.dat))	0.755	0.3
GetWaterSurface(CreateFromUrlLowRes(CTM42.dat))	0.755	0.3
GetWaterSurface(CreateFromUrlHighRes(CTM42.dat))	0.755	0.3

query. Therefore, any plans generating a water surface will be slightly penalized. Note that, while the variance is relatively high when compared with the user's expectations, the scores do not affect the user's expected overall ordering of the results. Although, it certainly can be posited that other properties, such as cost/quality of the workflow, can be factored into the relevance calculation.

5 Experimental Results

In this section, we discuss the performance evaluation for two aspects we described in Section 3, namely, (i) QoS handling and (ii) the benefits afforded by our caching framework in an actual cloud environment, particularly in conjunction with elasticity available in cloud environments. For both experiments, we use the coast line extraction query seen throughout this paper. We have evaluated these two features as we believe they are unique to the Auspice system. Referring back to Figure 3, the coast line workflow is composed of two services. (1) extractShoreline, which inputs a water level reading and a coastal terrain model (CTM) data file. (2) The parent service to extractShoreline is getWaterLevel, which inputs the time and location of interest from the user query. Whereas the water level service is negligible due to parallelism, the actual coast line extraction service is time consuming, i.e., CTM files are quite large.

5.1 QoS Handling

In this experiment, we allow the user to specify the amount accuracy they require, and report Auspice's efforts on meeting them. The execution time of this workflow can be shortened by sampling the CTM input file. However, as we illustrated in Section 3.1, the sampling rate of CTMs can take quite a departure from the domain-specific accuracies that the users specify. A model may predict actual errors (in meters) of the extractShoreline service operation on varying CTM sampling rates, but as for planning, Auspice must inversely determine the sampling rate on the input CTM from this model. Table 3 shows Auspice's efforts toward making these decisions on a particular CTM file.

The left half of the table are the correct correspondences between the sampling rate of the CTM (α%), and the physical errors predicted to have been induced by the loss in resolution. The right half the table shows Auspice's suggested α and the corresponding errors. Seen in the tables' juxtaposition, Auspice's automatic suggestions for α comes very close to the ideal values, and contributes insignificantly to the differences in physical errors. Although not shown here, the time taken for suggesting α is on the order of 10^{-6} seconds. Figure 7 displays the total execution times of the coast line extraction workflow on the user given accuracies (along the *x*-axis).

Table 3: Auspice Suggested Sampling Rates

Ideal		Suggested	
α%	Error (meters)	α%	Error (meters)
10	61.1441	10.00	61.1441
20	30.7205	19.93	30.7204
30	20.4803	29.91	20.4798
40	15.3603	39.89	15.3599
50	12.2882	49.87	12.2892
60	10.2402	59.98	10.2392
70	8.7773	69.88	8.7769
80	7.6801	79.90	7.6803
90	6.8268	89.94	6.8266
100	6.1441	100	6.1441

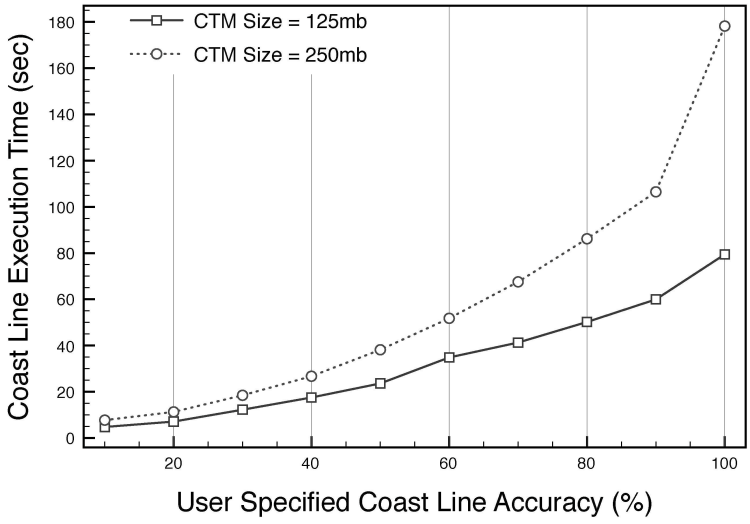


Fig. 7: QoS Handling in Coast Line Extraction

5.2 Caching in a Cloud Environment

We employ the Amazon Elastic Compute Cloud (EC2) [3] to support all of our experiments. Each Cloud node instance runs an Ubuntu Linux image on which our cache server logic is installed. Each image runs on a *Small EC2 Instance*, which, according to Amazon, comprises 1.7 GB of memory, 1 virtual core (equivalent to a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor) on a 32-bit platform. In all of our experiments, the caches are initially cold.

We executed repeated runs of the shoreline extraction query. The baseline execution time of the composite services, i.e., when executed without any caching, takes approximately ~ 23 seconds to complete. Because waiting for each request to complete over the lifetime of the experiment would take prohibitive amounts of time, we simulated its execution. The inputs to each shoreline extraction query consist the desired location and date. We have randomized these inputs over 64K possibilities for each request. The randomized query requests emulates the worst case scenario.

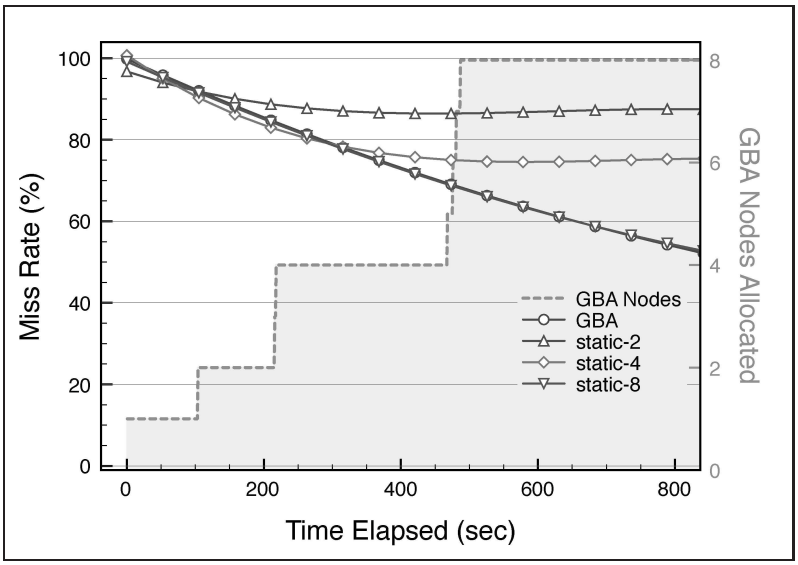
The initial experiment evaluates the effects of the cache without node contraction. In other words, the length of our eviction sliding window is ∞. Under this configuration, our cache is able to grow as large as it needs to handle the size of the cache. We run our cache system over static, fixed-node configurations (*static-2*, *static-4*, *static-8*). We then compare these static versions against our dynamic algorithm, Greedy Bucket Allocation (*GBA*), which runs over the EC2 public Cloud.

We executed the shoreline mashup at a rate of 50 query requests per second and 255 query requests per second. Figures 8a and 8b display the miss rates over the span of 800 seconds under these two configurations. Notice that the miss rates (against the left *y*-axis) for *static-2*, *static-4*, and *static-8* converge at relatively high values somewhat early into the experiment due to capacity being reached, although this behavior is exposed for *static-8* only in Figure 8b. Because we are executing *GBA* with an infinite eviction window, we do not encounter this capacity issue. In fact, our method continues to improve miss rates beyond the static versions, albeit requiring more nodes to handle the queries. Toward the end of the run, *GBA* is capable of attaining near-zero miss rates.

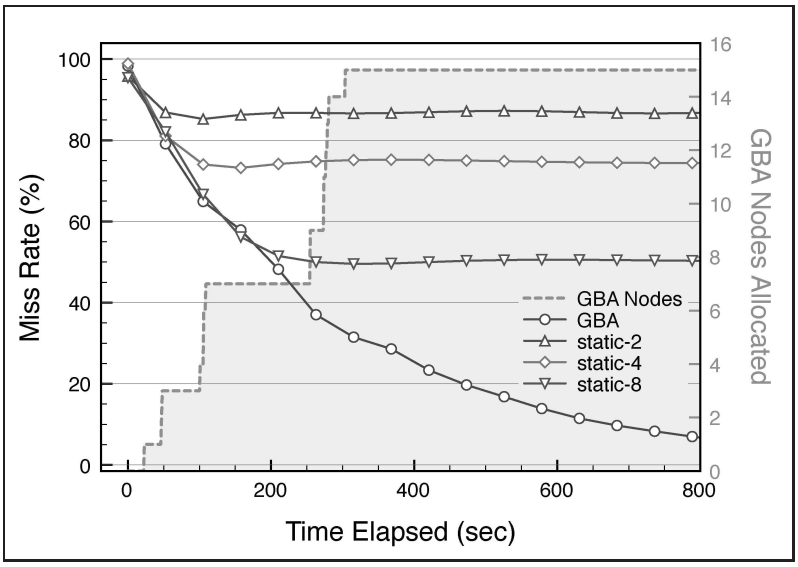
The node allocation behavior (against the right *y*-axis) shows that, over the lifespan of this experiment, *GBA* only employs a fraction of the nodes needed to perform better than *static-8*. In fact, an average of $\lceil 5.557 \rceil = 6$ nodes is used over the query rate = 50 experiment and an average of $\lceil 12.6 \rceil = 13$ nodes in the query rate = 255 version. This translates to less overall EC2 usage cost per performance over static allocations. The growth of nodes is also not unexpected, though, at first glance it appears to be exponential. Node increases are concentrated toward the beginning of execution because the overall capacity is too small to handle the query rate.

Figures 9a and 9b, which show the respective mean query execution times, corresponds directly to the above miss rates. To create these figures, for each second elapsed in our execution, we averaged the query execution times (over 50 and 255 respectively) and plotted them (against the left *y*-axis). The speedup provided by the static versions expectedly flatten somewhat quickly, again, due to the nodes reaching capacity. *GBA*, on the other hand, performed better, but requiring far less nodes throughout the length of the experiment.

To show node allocation and migration overhead, in Figures 10a and 10b, the maximum query execution time (*GBA-max*) is displayed for *GBA*. The spikes in the execution time expectedly align with EC2 instance allocation which, in our experience, can take extensive amounts of time. We posit that the demand for node allocation diminishes as the experiment proceeds even with high querying rates. But while node allocation overheads are high, its negative impact on overall speedup is amortized over the span of the experiment because it is only seldom invoked. Moreover,

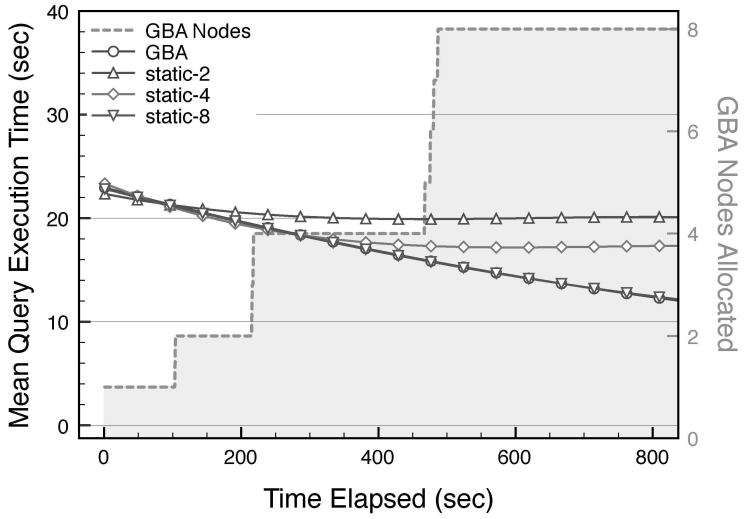


(a) Querying Rate = 50/sec

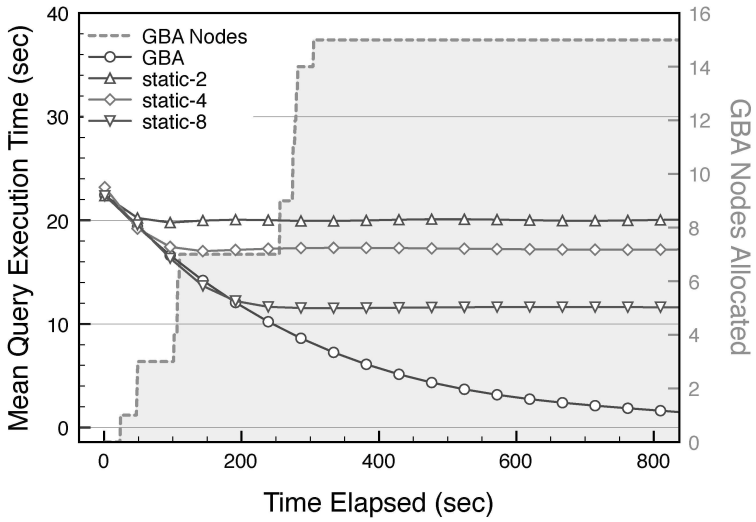


(b) Querying Rate = 255/sec

Fig. 8: Miss Rate

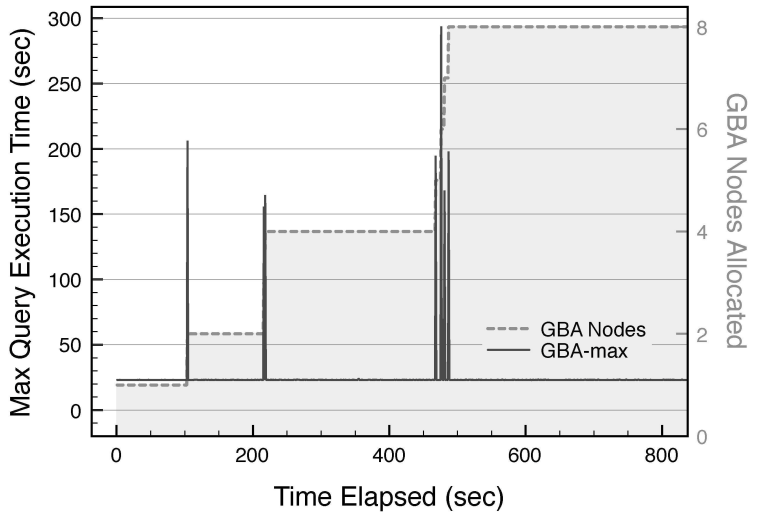


(a) Querying Rate = 50/sec

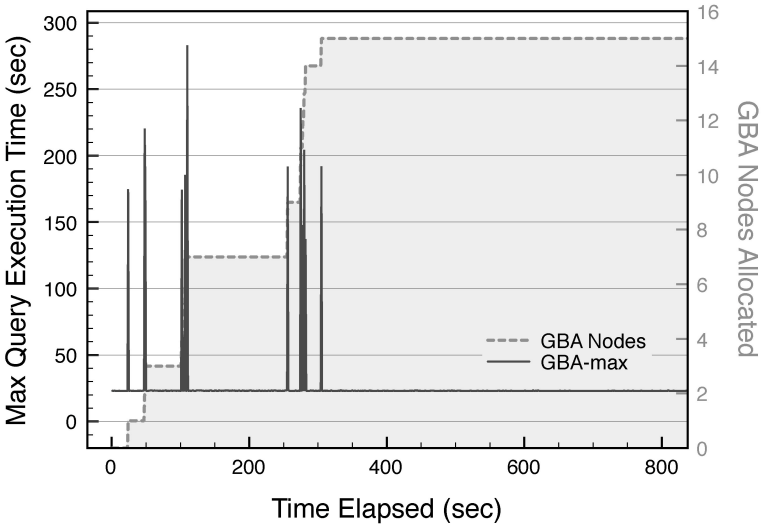


(b) Querying Rate = 255/sec

Fig. 9: Mean Query Times



(a) Querying Rate = 50/sec



(b) Querying Rate = 255/sec

Fig. 10: Max Query Times for *GBA*

techniques, such as preloading EC2 instances, can also be used to further minimize this overhead although these have not been considered in this paper.

Next, we evaluated the sliding window approach of maintaining our cache. Two separate experiments were devised to show the effectiveness of the sliding window and to show that our cache is capable of scaling down. Again, we randomize the query inputs over 64K possibilities. We begin these experiments with a querying rate of 25/sec. Then between 100 and 300 seconds, the querying rate is increased to 60/sec and input possibilities are decreased down to 32K. This emulates a period of frequent and highly related queries being issued. After 300 seconds into the experiment, the querying rate resumes back to 25/sec with 64K possible inputs.

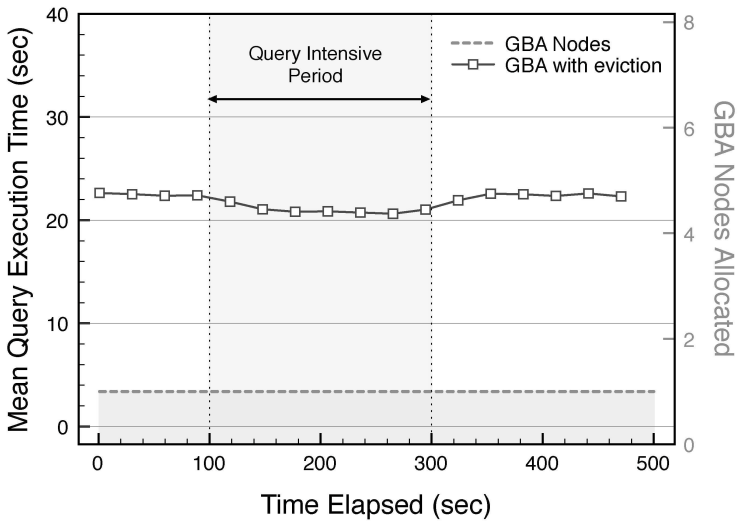
Figures 11a and 11b shows the results of this experiment for sliding window sizes of 50 sec and 100 sec respectively. Specifically, the cache will attempt to maintain, with high probability, all records that were queried in the most recent 50 and 100 seconds. The decay, α, has been fixed at 0.99 for these experiments. The eviction threshold, T_λ, is set at the baseline $\alpha^{m-1} \approx 0.367$ to evict any key which had been only queried within the evicted slice.

As demonstrated by these experiments, our cache adapts to the query intensive period by lowering mean query execution times. We can also observe that, after the query intensive period expires at 300 seconds, the sliding window detects the normal querying rates and removes nodes as they become superfluous (though this is only noticeable in Figure 11b. Here, the nodes do not decrease back down to 1 because our contraction algorithm is slightly conservative: Recall that we only contract if two of the lowest loaded nodes account for less than half the space required to store their merged cache.

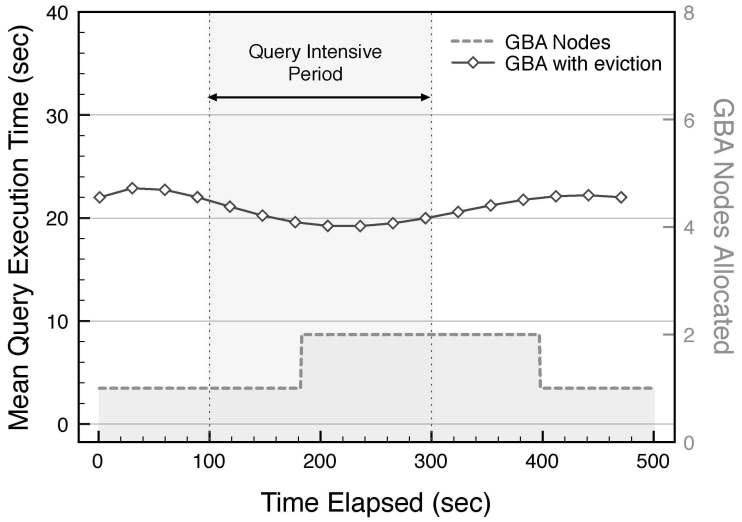
The differences between the two side-by-side figures also imply that the size of the sliding window is a determinant factor on both performance and cost. Since the querying rate and sliding window are so small, this version never requires more than 1 node, and it requires less nodes than its counterpart in Figure 11b. However, the tradeoff is clear in that it cannot reap the benefits of the larger cache afforded by the longer sliding window. The same experiments were conducted for higher querying rates to produce Figures 12a and 12b. We increased the normal querying rate to 50/sec and intensive querying rate to 255/sec. As seen in the figures, the same cache elasticity behavior can be expected upon these very high rates of querying, showing us some positive results on scalability. In fact, in terms of performance our system welcomes higher querying rates, as it populates our cache more frequently within the sliding window.

6 Related Works

Among the first systems to utilize workflows to manage scientific processes is ZOO [32], which employs an object-oriented language to model the invocation of processes and the relationships between them. Another notable system, Condor [37], was originally proposed to harvest the potential of idle CPU cycles. Soon after, de-

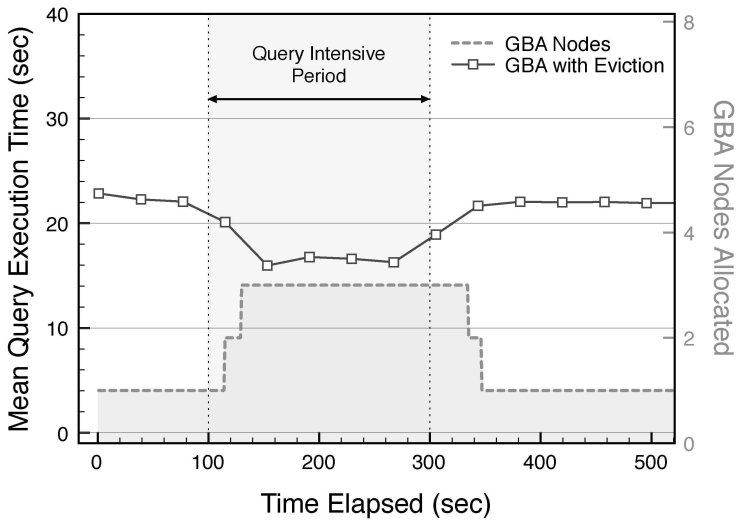


(a) Sliding Window Size = 50 sec

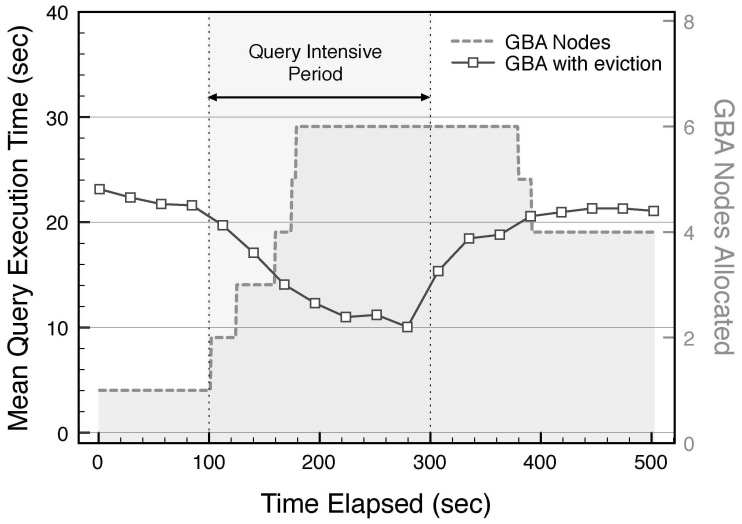


(b) Sliding Window Size = 100 sec

Fig. 11: Cache Contraction (Normal Query Rate = 25/sec, Intensive Rate = 60/sec)



(a) Sliding Window Size = 50 sec



(b) Sliding Window Size = 100 sec

Fig. 12: Cache Contraction (Normal Query Rate = 50/sec, Intensive Rate = 255/sec)

pendent processes (in the form of directed acyclic graphs), were being scheduled on Condor systems using DAGMan [17]. Recently, with the onset of the Data Grid, Condor has been integrated with the Globus Tookit [24] into Condor-G [25]. Pegasus [19] creates workflow plans in the form of Condor DAGMan files, which then uses the DAGMan scheduler for execution.

Many systems currently allow users to guide workflow designs. In Casati et al.'s eFlow [10], a workflow's structure is defined by users, but the instantiation of services within the process is dynamically allocated by the eFlow engine. Sirin et al. proposed a user interactive composer that provides semi-automatic composition [42]. In their system, after each time that a particular service is selected for use in the composition, the user is presented a filtered list of possible choices for the next step. This filtering process is made possible by associating semantic data with each service. To complement the growing need for interoperability and accessibility, many prominent workflow managers, including Pegasus [19], Taverna [40], Kepler [2] (the service-enabled successor to the actor and component-based Ptolemy II [9]), and Triana [38] have become attuned with service-oriented systems.

Workflow systems with QoS support have also been developed. For example, Askalon offers system-level QoS, e.g., throughput and transfer rates [7, 8]. Glatard's service scheduler exploits parallelism within service and data components [31]. Lera et al. have developed a performance ontology for dynamic QoS assessment [36]. Kumar et al. [35] have developed a framework for application-level QoS support. Their system, which integrates well-known Grid utilities (the Condor scheduler [25], Pegasus [19], and DataCutter [6], a tool which enables pipelined execution of data streams) considers *quality-preserving* (e.g., chunk size manipulation, which does not adversely affect accuracy of workflow derivations) and *quality-trading* QoS parameters (e.g., resolution, which could affect one QoS in order to optimize another). In quality-preserving support, the framework allows for parameters, such as chunk-size, to be issued. These types of parameters have the ability to modify a workflow's parallelism and granularity, which potentially reduces execution times without performance tradeoffs. For quality-trading QoS support, an extension to the Condor scheduler implements the tradeoff between derived data accuracy for improved execution time. We believe that Auspice is the first to propose algorithms reconciling the tradeoffs between data reduction parameters against actual domain errors within the scientific application.

7 Conclusion

At the Ohio State University, we have developed Auspice, an automatic service composition engine that supports several functionalities, outlined below.

- Enables a nonintrusive framework for sharing scientific data sets and Web services through metadata indexing.
- Composes known services and data sets in disparate ways to derive user queries.

- Through error models, it adaptively controls tradeoffs between execution time and application error to meet QoS constraints.
- Caches intermediate results for trivializing future service invocations.

We are currently exploring several directions to further our system's functionalities. We will finalize the keyword search process, and also, inspect methods toward the seamless integration of Deep Web data sources. We also believe that the emergence of the Cloud affords us excellent opportunities to develop fruitful research. On this front propose to examine the effects of parallelism in a Cloud versus static systems, such as a cluster. It is well known that one way to reduce execution time is by parallelizing large, independent computations within scientific workflow systems [35]. While it is tempting to maximize the usage of the Cloud's "infinite resources," overheads, such as deploying virtual machine images onto new nodes exist. For disparate applications and data sets, there will certainly be cases where the Cloud's overheads are amortized against the execution time speed ups. Conversely, there will also be applications where the Cloud's overheads override the benefits for its use. We believe that a study on workloads, parallelism granularity, and cost would be beneficial toward understanding the tradeoffs between Cloud usage and other preexisting infrastructures.

Acknowledgments

This work is supported by NSF grants 0541058, 0619041, and 0833101. The equipment used for the experiments reported here was purchased under the grant 0403342.

References

1. S. Agrawal. Dbxplorer: A system for keyword-based search over relational databases. In *In ICDE*, pages 5–16, 2002.
2. I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows, 2004.
3. Amazon elastic compute cloud, http://aws.amazon.com/ec2.
4. M. Armbrust, *et al.* Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.
5. The atlas experiment, http://atlasexperiment.org.
6. M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed processing of very large datasets with datacutter. *Parallel Computing*, 27(11):1457–1478, Novembro 2001.
7. I. Brandic, S. Pllana, and S. Benkner. An approach for the high-level specification of qos-aware grid workflows considering location affinity. *Sci. Program.*, 14(3,4):231–250, 2006.
8. I. Brandic, S. Pllana, and S. Benkner. Specification, planning, and execution of qos-aware grid workflows within the amadeus environment. *Concurr. Comput. : Pract. Exper.*, 20(4):331–345, 2008.

9. C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous concurrent modeling and design in java (volume 2: Ptolemy ii software architecture). Technical Report 22, EECS Dept., UC Berkeley, July 2005.
10. F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eFlow. In *Conference on Advanced Information Systems Engineering*, pages 13–31, 2000.
11. D. Chiu and G. Agrawal. Enabling ad hoc queries over low-level scientific datasets. In *Proceedings of the 21th International Conference on Scientific and Statistical Database Management (SSDBM'09)*, 2009.
12. D. Chiu and G. Agrawal. Flexible caches for derived scientific data over cloud environments. Technical Report OSU-CISRC-7/09-TR35, Department of Computer Science and Engineering, The Ohio State University, July 2009.
13. D. Chiu and G. Agrawal. Hierarchical caches for grid workflows. In *Proceedings of the 9th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. IEEE, 2009.
14. D. Chiu, S. Deshpande, G. Agrawal, and R. Li. Composing geoinformatics workflows with user preferences. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'08)*, New York, NY, USA, 2008.
15. D. Chiu, S. Deshpande, G. Agrawal, and R. Li. Cost and accuracy sensitive dynamic workflow composition over grid environments. In *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (Grid'08)*, 2008.
16. D. Chiu, S. Deshpande, G. Agrawal, and R. Li. A dynamic approach toward qos-aware service workflow composition. In *Proceedings of the 7th IEEE International Conference on Web Services (ICWS'09)*. IEEE Computer Society, 2009.
17. Condor dagman, http://www.cs.wisc.edu/condor/dagman.
18. Dublin core metadata element set, version 1.1, 2008.
19. E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. C. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.
20. C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. The MIT Press, 1998.
21. Metadata ad hoc working group. content standard for digital geospatial metadata, 1998.
22. Federal geospatial data clearinghouse, http://clearinghouse.fgdc.gov.
23. I. Foster. Service-oriented science. *Science*, 308(5723):814–817, May 2005.
24. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.
25. J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 7–9, San Francisco, California, August 2001.
26. gbio: Grid for bioinformatics, http://gbio-pbil.ibcp.fr.
27. Bioinfogrid, http://www.bioinfogrid.eu.
28. Biomedical informatics research network, http://www.nbirn.net.
29. Cyberstructure for the geosciences, http://www.geongrid.org.
30. The geography network, http://www.geographynetwork.com.
31. T. Glatard, J. Montagnat, and X. Pennec. Efficient services composition for grid-enabled data-intensive applications. 2006.
32. Y. E. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti. Zoo: A desktop experiment management environment. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 274–285, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.
33. C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient b+tree-based indexing of moving objects. In *Proceedings of Very Large Databases (VLDB)*, pages 768–779, 2004.
34. D. Karger, *et al.* Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, 1997.

35. V. S. Kumar, P. Sadayappan, G. Mehta, K. Vahi, E. Deelman, V. Ratnakar, J. Kim, Y. Gil, M. Hall, T. Kurc, and J. Saltz. An integrated framework for performance-based optimization of scientific workflows. In *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 177–186, New York, NY, USA, 2009. ACM.
36. I. Lera, C. Juiz, and R. Puigjaner. Performance-related ontologies and semantic web applications for on-line performance assessment intelligent systems. *Sci. Comput. Program.*, 61(1):27–37, 2006.
37. M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.
38. S. Majithia, M. S. Shields, I. J. Taylor, and I. Wang. Triana: A Graphical Web Service Composition and Execution Toolkit. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 514–524. IEEE Computer Society, 2004.
39. Biological data working group. biological data profile, 1999.
40. T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.
41. Sloan digital sky survey, http://www.sdss.org.
42. E. Sirin, B. Parsia, and J. Hendler. Filtering and selecting semantic web services with interactive composition techniques. *IEEE Intelligent Systems*, 19(4):42–49, 2004.
43. V. H. University and V. Hristidis. Discover: Keyword search in relational databases. In *In VLDB*, pages 670–681, 2002.
44. M. Wan, A. Rajasekar, R. Moore, and P. Andrews. A simple mass storage system for the srb data grid. In *MSS '03: Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)*, page 20, Washington, DC, USA, 2003. IEEE Computer Society.
45. Extensible markup language (xml) 1.1 (second edition).

Parameter sweep job submission to Clouds

P. Kacsuk, A. Marosi, M. Kozlovszky, S. Ács, Z. Farkas

Abstract This chapter introduces the existing connectivity and interoperability issues of Clouds, Grids and Clusters, and provides solutions to overcome these issues. The paper explains the principles of parameter sweep job execution by P-GRADE portal and gives some details on the concept of parameter sweep job submission to various Grids by the 3G Bridge. Then it proposes several possible solution variants how to extend the parameter sweep job submission mechanism of P-GRADE and 3G Bridge towards Cloud systems. Finally, it shows the results of performance measurements that were gained for the proposed solution variants.

1 Introduction

The e-science infrastructure eco-system has been recently enriched with Clouds and as a result the main pillars of this eco-system are:

- Supercomputer-based grids (e.g. DEISA, TeraGrid, etc.)
- Cluster-based Grids (e.g. EGEE, NorduGrid, OSG, SEE-GRID, etc.)
- Desktop Grids (e.g. BOINC-based, XtremWeb-based, etc.)
- Clouds (e.g. Eucalyptus, OpenNebula, Amazon, etc.)

All pillars have their own advantages that make them attractive for a certain application area. Supercomputers are very reliable, robust services that can be used to solve tightly-coupled compute- or data-intensive applications. Their drawback is the high investment and maintenance cost that is affordable only for a small number of distinguished computing centers that typically receive financial support from national or regional governments. Organizing them into a Grid system where large number of scientists can access them in a balanced and managed way significantly increases their utilization.

P. Kacsuk, A. Marosi, M. Kozlovszky, S. Ács, Z. Farkas
MTA SZTAKI, Hungary-1518 Budapest. P.O. Box 63., e-mail: kacsuk@sztaki.hu

Cluster-based service Grids are less expensive than Supercomputer-based grids. They are very flexible in the sense that they can efficiently run any kind of applications including tightly- and loosely-coupled, compute- and data-intensive applications. These are the most popular forms of building Grid systems. Nowadays Grid computing is used in many research domains to provide the required large set of computing and storage resources for e-scientists. Europe's leading Grid computing project Enabling Grids for E-sciencE (EGEE) is a good example, providing a computing support infrastructure for over 13,000 researchers world-wide, from fields as diverse as high energy physics, chemistry, engineering, earth and life sciences.

Desktop Grids represent the least expensive form of collecting resources. The two main options are the global volunteer computing systems and the institutional Desktop Grid (DG) systems. In the global version the spare cycles of typically home computers are donated on a volunteer basis. In DG systems, anyone can bring resources into the grid system, offering them for the common goal of that Grid. Installation and maintenance of the grid resource middleware is extremely simple, requiring no special expertise. This ease of use enables large numbers of donors to contribute into the pool of shared resources. In the institutional DG systems the spare cycles of the existing computers of an (academic or commercial) institution are exploited. Since it uses the free cycles of already available computers, it needs only minimal initial investment and maintenance cost. Volunteer DG systems are very popular and collect very large number of resources in the range of 10K – 1M CPUs. The most well-known example of an application that has successfully adapted to DGs is the SETI@HOME project, in which approximately four million PCs have been involved. However, the drawback of Desktop Grids is that they are not suitable for all kind of applications. They can efficiently support only bag of task (parameter sweep and master/worker) compute-intensive applications, however they can not meet strict QoS and SLA requirements.

The recently emerged Cloud systems can be used when strict QoS and SLA requirements are applied. In many cases resource requirements are higher than the available Grid infrastructure resources can provide. It is very common, that researchers need shorter response time, and reliable infrastructure with strict SLAs. Cloud based infrastructure can fulfill such requirements in many cases however, the porting costs of complex research application is hard to finance.

Unfortunately, in many cases these pillars are separated from each other and cannot be used simultaneously by the same e-scientist to solve a large-scale single application. Partial results of interconnecting these systems have been achieved in the past. Cluster and supercomputer based grids can be referred to as Service Grids (SG) since they provide managed cluster and supercomputer resources as 7/24 services. The OGF PGI (Production Grid Infrastructure) working group has put significant effort to solve the interoperability problem of production service grids and yet even the interoperation of various cluster-based and supercomputer-based grids is not fully solved. The recently formed EMI (European Middleware Initiative) project aims at integrating the three major European grid middleware systems (ARC, gLite, Unicore) into a unified middleware distribution (UMD). If we could provide seamless interoperability and easy application migration between Grid and Cloud infras-

tructure, we can enable the migration of existing scientific applications from Grids towards Clouds and we can support the better requirements matching between applications and infrastructures. The commercial Cloud infrastructure can support large scale resource requirements in a reliable way. Solving the interoperability issues between Grids and Clouds, Grid researchers can use in a dynamic way additional Cloud resources if they have exhausted their available Grid resources.

BOINC-based and XtremWeb-based DG systems have been successfully integrated with gLite-based service grids within the EDGeS project [1] at two levels. At the middleware level a bridge technology was developed called as 3G Bridge that enables a seamless extension of gLite-based SGs with DGs and vice versa gLite-based SGs can support DG systems in a seamless way. At the application level the Grid execution back-end of P-GRADE grid portal was extended with the capability of submitting predefined applications not only into service grids but also to Desktop Grids.

The usage scenario investigated in EDGeS was as follows. Using the P-GRADE grid portal a user has prepared a large workflow application where some of the workflow nodes require parameter sweep execution with many different parameter sets. In such case the user can direct the execution of such node to a DG system that already supports the application represented by this workflow node. Other nodes of the workflow can be directed to local resources or SG resources as defined by the user. In this way the user can direct different parts of the workflows to the cheapest available Grid resources.

The objective of the research reported in this chapter is to investigate how the application level approach of the EDGeS project can be extended for clouds, i.e., how to direct the execution of the parameter sweep workflow nodes of a P-GRADE workflow not only to DG systems but also to Clouds especially when SLA requirements are more strict than a DG system can satisfy.

The chapter is divided into 6 sections. After this short introduction Section 2 explains the principles of parameter sweep job execution by P-GRADE portal. The next section gives some details on the concept of parameter sweep job submission to various Grids by the 3G Bridge. Section 4 introduces several possible solution variants for parameter sweep job submission to Cloud systems. Section 5 shows the results of performance measurements. Finally, Section 6 details some related research results.

2 Principles of parameter sweep job submission by P-GRADE portal

In the academic world science gateways gain more and more popularity especially for developing and running large scale applications. In Europe one of the most popular generic purpose science gateway is the P-GRADE portal. The basic concepts of the P-GRADE Portal based science gateway solution were mainly developed during the grid era. Therefore P-GRADE portal is used by many national grids (UK NGS,

Grid Ireland, Belgium Grid, SwissGrid, Turkish Grid, Hungarian Grid, Croatian Grid), by several regional grids (South-East European Grid, Baltic Grid, UK White Rose Grid) and by several science specific virtual organizations (Chemistry Grid, Economy Grid, Math Grid, etc.). In recent years P-GRADE portal became popular even outside Europe: in Grid Malaysia established by MIMOS Berhad, Grid Kazakhstan, and Armenian Grid. In total thousands of computers located in more than 45 countries can be accessed via the P-GRADE portal solution.

P-GRADE portal (Parallel Grid Run-time and Application Development Environment) [2] [3] is an open source tool set consisting of a service-rich, workflow-oriented graphical front-end and a back-end enabling execution in various types of Cluster/Grid/Cloud systems. It supports workflows composed of sequential jobs, parallel jobs and application services. P-GRADE portal hides the complexity of infrastructure middleware through its high-level graphical web interface, and it can be used to develop, execute and monitor workflow applications on SG systems /built with Globus, EGEE (LCG or gLite), ARC/, on Clusters /using PBS, LSF/, and on DG system /using BOINC and XtremWeb middleware/. P-GRADE portal installations typically provide the user with access to several middleware technologies, using a single login.

The main features of P-GRADE Portal are:

- Seamless interoperability with the widest range of technologies and Cluster/S-G/DG middleware (Globus Toolkit 2, Globus Toolkit 4, LCG, gLite, ARC, BOINC, PBS, LSF, BOINC, XtremWeb) among the available portal/gateway solutions.
- Multi-grid access mechanism to simultaneously utilize multiple grid implementations [2].
- Fully compliant with all the security features used in Grids (X.509 certificates, proxy credentials, etc.),and support of other authentication solutions (e.g.: Sibboleth).
- Support of workflow-based application design with built-in graphical editor
- Monitoring, accounting and visualization, quota management facilities of infrastructure.
- Legacy application publish and reuse capabilities by the GEMLCA mechanism.
- Large international developer community, world-wide usage with a high number of distributed large-scale, parallel, scientific applications.

The P-GRADE portal applies a DAG (directed acyclic graph) based workflow concept (shown in Fig. 1). In a generic workflow, nodes (shown in Fig. 1 as large squares) represent jobs, which are basically batch programs to be executed on a computing element. Ports (shown here as small squares around the large ones) represent input/output files the jobs receiving or producing. Arcs between ports represent file transfer operations. The basic semantics of the DAG-based workflow is that a job can be executed if and only if all of its input files are available. This semantics is enforced by the Condor DAGMan workflow manager that is used internally in P-GRADE portal.

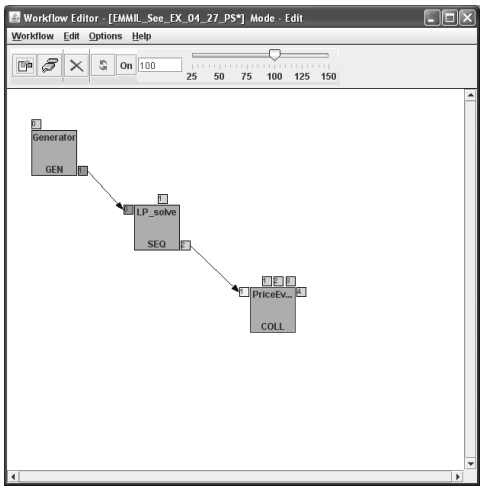


Fig. 1 Example workflow in P-GRADE Portal

In our experience, user communities have shown substantial interest in being able to run programs parallel with different input files. P-GRADE portal supports this kind of parallelization called Parameter Sweep, or Parameter Study at a high level. The original "job" idea has been extended by two special jobs called Generator and Collector to facilitate the development of parameter sweep (PS) type workflows in P-GRADE portal. The Generator job is used to generate the input files for all parallel jobs automatically (called Automatic Parameter Input Generator) or by a user-uploaded application (called Normal Generator). The Collector will run after all parallel executions are completed and then collects all parallel outputs [4]. All jobs connected to a Generator will run in as many instances as many input files are generated by the Generator (shown in Fig. 2).

3 Principles of parameter sweep job submission to various Grids by 3G Bridge

Originally, P-GRADE portal supported parameter sweep job submission to Globus and gLite based Service Grids. However, for compute-intensive parameter sweep jobs Desktop Grids are more ideal than SGs since they are less expensive. Directing these kinds of parameter sweep jobs into DGs will enable to use the expensive service Grid resources for other type of applications that are not supported by DGs.

In order to direct parameter sweep jobs to Desktop Grids the 3G Bridge service developed in the EDGeS project has been interfaced with P-GRADE portal. The internal architecture of the 3G Bridge can be seen in Fig. 3 where the following components can be identified:

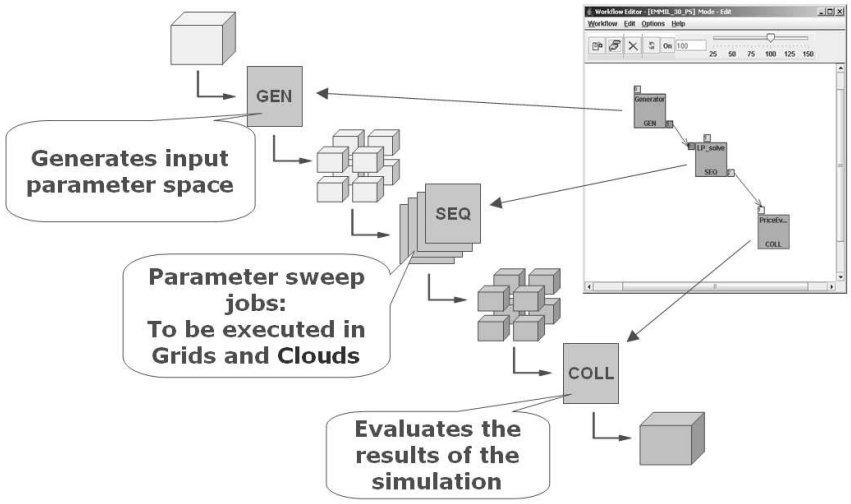


Fig. 2: Parameter sweep solution in P-GRADE Portal with Generator and Collector

- WSSubmitter provides a web service interface in order to access the 3G Bridge services as a usual web service.
- HTTPD enables to download files from the 3G Bridge machine.
- Job Database is used for storing the jobs to be executed in the target grids.
- Queue Manager is responsible for handling the jobs in the database by using a very simple scheduler for calling the specific plug-ins.
- Grid Handler Interface provides a generic interface above the grid plug-ins.
- Plug-ins responsible for submitting the jobs into different destination infrastructures.
- Download Manager is an internal component of the 3G Bridge that downloads job input files from 3G Bridge clients.

Whenever a new destination infrastructure is to be connected to P-GRADE portal the corresponding plug-in should be written and attached to the Grid Handler Interface. In the EDGeS project the gLite, XtremWeb, BES and BOINC plug-ins have been developed. With these plug-ins the user can submit PS jobs to gLite, XtremWeb, ARC and UNICORE, or BOINC Grids.

Once a job should be submitted using the 3G Bridge, P-GRADE Portal runtime makes use of the WSClient application, the client of WSSubmitter. Using this tool, P-GRADE portal can send jobs to the target 3G Bridge service and plugin. The jobs' input files are sent along with the job submission request, so in this scenario the Download Manager has no real task. Once a job has been submitted, P-GRADE portal periodically updates the job's status using the WSClient application. Finally, if the job has finished, P-GRADE portal fetches the output files from the 3G Bridge service machine using the HTTPD server.

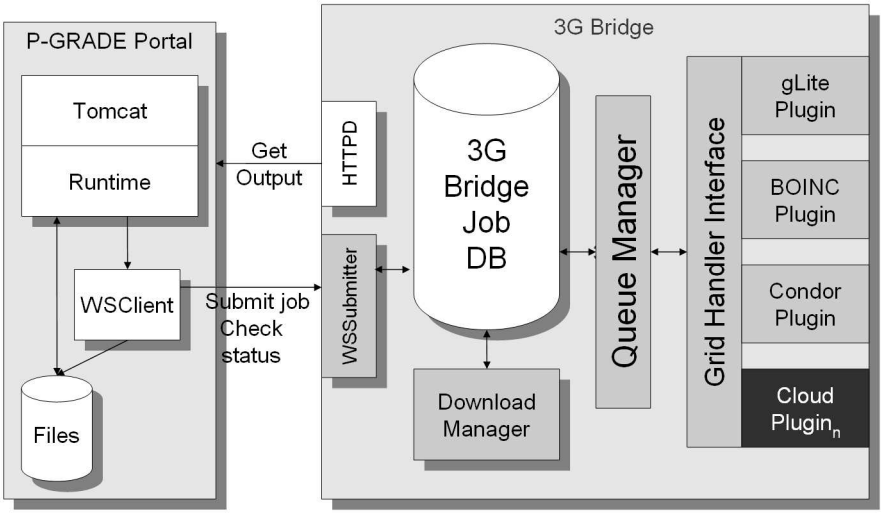


Fig. 3: Interfacing P-GRADE portal and 3G Bridge

DC-API [5] is used to implement both the BOINC and Condor plug-ins [6]. It provides a simple uniform API for writing master-worker type applications for different Grids (currently BOINC, Condor and the Hungarian Cluster Grid are supported). This means applications using DC-API do not need modifications when moving from one supported platform to another, just re-linking with the appropriate DC-API backend library. The Condor and BOINC 3G-Bridge plug-ins are basically instances of the same "DC-API-Single" plug-in, but linked with a different DC-API backend library. This also means that the plug-ins are interchangeable, instead of a Condor plug-in a BOINC one could be used (with some restrictions).

4 Variants of creating 3G Bridge cloud plug-ins

In order to submit PS jobs not only to grids but also to Clouds the possible variants of creating a Cloud plug-in for 3G Bridge should be investigated. Once the Cloud plug-in is available the portal user can submit PS jobs not only to SGs and DGs but also to Clouds. The Cloud plug-in has to solve three problems to submit and manage PS jobs in Cloud resources:

- Cloud resource management: takes care of allocating Cloud resource for PS jobs when they arrive and removing the Cloud resources when no PS jobs are available for Cloud execution.
- Job submission: Submits PS jobs waiting in the 3G Bridge Job Database to the allocated Cloud resources.

- Job scheduling on Cloud resources: Decides which allocated Cloud resources to be used by the individual PS jobs.

4.1 The naive solution

A naive solution would be that for each incoming PS job the 3G Bridge plug-in allocates a new Cloud resource and submit the job to this cloud resource. Once the execution of the job on the Cloud resource is finished the 3G Bridge plug-in removes the cloud resource. This is obviously a very non-optimal solution that assumes that the number of available Cloud resources is unlimited as well as the budget the user can spend on them. In a realistic scenario the user can afford only a certain number of resources and once those resources are loaded with jobs some intelligent job scheduling decision is needed to which already loaded resource to submit the newly arriving PS job. Therefore, more sophisticated solutions have to be investigated where the number of usable Cloud resources has an upper limit. We have considered three basic solution variants:

- Variant 1: Independent Cloud resources with local job managers
- Variant 2: Communicating Cloud resources with centralized job manager
- Variant 3: Independent Cloud resources with centralized job managers

In all three variants we need a job manager that can realize job scheduling on the finite set of resources. Any available job manager system can be considered for this purpose. We selected Condor since it is widely used in the academic community.

4.2 Independent cloud resources with local job managers

The concept of the first variant is shown in Fig. 4. The main idea is that on each Cloud resource the 3G Bridge Cloud plug-in deploys a VM image containing a Condor job manager and a Condor worker. The Condor job manager takes care of scheduling the jobs submitted to this resource while the worker will execute them. Due to the Condor job manager many jobs can be sent to the same Cloud resource. In the 3G Bridge Cloud plug-in various scheduling algorithms can be applied to select the most optimal Cloud resource. For example, a simple algorithm that realizes both scheduling and Cloud resource management can be the following where the decision is based on the number of jobs allocated for the Cloud resources:

According to this algorithm the 3G Bridge Cloud plug-in is going to start a new instance if there is no free instance and the number of running instances is less than the upper limit of the usable instances, and in this case the job will be submitted to this new instance. Otherwise the new job will be submitted to that instance that has the minimum number of assigned jobs. Notice that this algorithm assumes that the execution time of the PS jobs are approximately equal. It is true for many PS

Start and submit:

1: **if** $m = 0 \wedge k < n$ **then**
2: $instanceid \leftarrow start\_instance()$
3: **else**
4: $instanceid \leftarrow find\_min(j_0..j_{k-1})$
5: **end if**
6: submit_job(instanceid, jobid)

Stop:

1: **for** $i \leftarrow 0$ to $k-1$ **do**
2: **if** $t_i < timestamp(\texttt{"-20 minutes"}) \wedge j_i = 0$ **then**
3: stop_instance(i)
4: **end if**
5: **end for**

Legend:

m: number of free instances
k: total number of running instances
n: maximum number of instances
j_x: number of jobs queued on instance x
t_x: timestamp of last execution on instance x

applications but not always. If this assumption does not hold the load of the different Cloud resources could significantly vary and there is no remedy for such problems in this architecture.

The Cloud resource management part of the algorithm will stop an instance if its job queue is empty and there was no activity in the last 20 minutes. This 20 minutes buffer time is used to avoid the frequent and useless stopping/restarting activities when the time between the incoming jobs are in the range of several minutes.

The 3G Bridge Cloud plug-in is responsible to allocate the required number of Condor job manager/worker instances in the cloud and submit the PS jobs to the Condor job managers of the instances. The decision to which Condor instance a certain PS job should be sent is also taken by the 3G Bridge Cloud plug-in. It means that all the three functions (cloud resource management, job submission, job scheduling) should be performed by the 3G Bridge Cloud plug-in. Integrating these three functions into one monolithic plug-in is not a desirable solution since it means that every time a new type of Cloud is to be connected to 3G Bridge, a new complex plug-in with all these functionalities should be developed. A better approach would be to develop an independent simple plug-in for all these three functionalities.

As a summary one can say that the advantages of this variant are as follows:

- Still simple to implement compared to naive solution.
- Uses reliable method for task submission (e.g. Condor WS API).
- The number of used Cloud resources is controllable.

However, there are several serious drawbacks as well:

- A single plug-in manages both the resource allocation, job submission and job scheduling that requires major redesign whenever new Clouds (e.g. OpenNebula, Eucalyptus, etc.) or new job managers (PBS, LSF, etc.) are to be used.
- Special prepared VM image is required.
- Since every VM image contains a Job Manager it raises some overhead. E.g.: in case of Condor it is around 200Mb which has extra cost both in the storage area

and in the communication time when the image is moved from the storage to the resources.

- Nodes are separated from each other and hence there is no way to reschedule jobs from a node to another. As a consequence some nodes might become over-committed while others are empty.

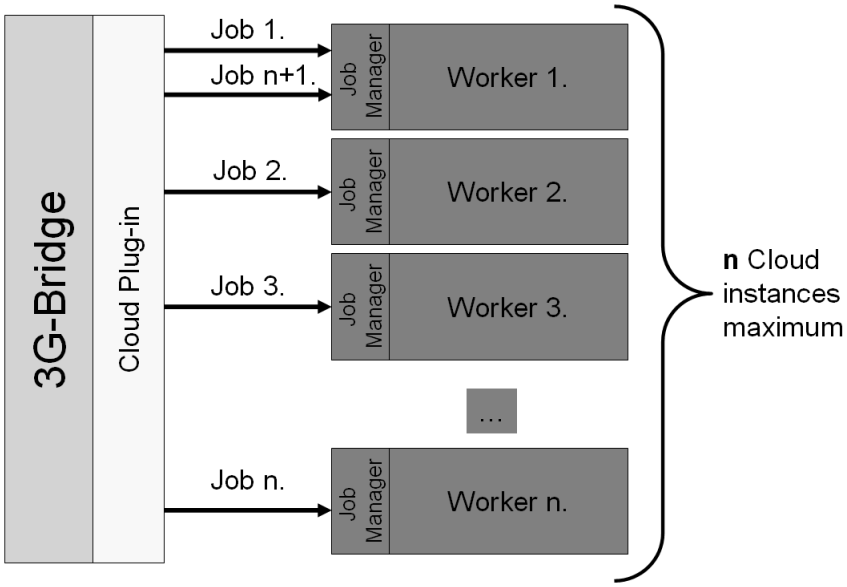


Fig. 4: Variant 1: Independent cloud resources with local job managers

4.3 Communicating cloud resources with centralized job manager

The architecture and concept of the second variant is shown in Fig. 5. The main difference compared with the first variant is that the Condor job manager is placed in a separate image. The other VM images contain only the Condor worker, and the Cloud resources where they are deployed should be able to communicate with the Cloud resource of the Condor job manager.

The 3G Bridge Cloud plug-in becomes significantly simpler since there is no need to schedule the incoming PS jobs among the Condor worker cloud resources. This scheduling will be done by the Condor job manager. The plug-in needs only to start/stop instances for the job manager and submit the jobs to the Condor job manager. The resource management algorithm of the 3G Bridge Cloud plug-in is as follows:

Start and submit:

```
1: if tj/k > q ∧ k < n then
2:     start_worker()
3: end if
4: submit_job_to_frontend(jobid)
```

Stop:

```
1: for i ← 0 to k − 1 do
2:     if tj/(k − 1) < q ∧ t_i <
       timestamp("-20 minutes") then
3:         stop_worker(i)
4:     end if
5: end for
```

Legend:

m: number of free instances
k: total number of running instances
n: maximum number of instances
t: timestamp of last job submission
tj: total number of jobs in the Cloud
q: preferred maximum job number per worker

According to this algorithm the 3G Bridge cloud plug-in will deploy a new instance only if the average job number per worker surpasses the preferred maximum job number per worker and the number of running instances is less the permitted, maximum number of instances. A worker Cloud resource will be stopped if there was no job submission in the last 20 minutes and the average job number per worker is less than the preferred maximum job number per worker. A worker will be selected based on its "shutdown time window". This can mean different things: Commercial Cloud providers (e.g. Amazon EC2) may charge for instance hours, rounded up, thus an instance will be charged for an hour regardless if it was running for 15 or 59 minutes. In this case the shutdown time window could be open in the 54-58th minutes of every hour in the lifetime of an instance and the plug-in is allowed to shutdown the instance during this time window only.

The advantage of this architecture is that the job manager balances tasks between the Cloud resources. As a result no resource gets overcommitted while others starve. Further advantage is that parallel (MPI, PVM, Map-Reduce, Hadoop) applications can also be executed in this architecture. However, there are several serious drawbacks as well. Two specially prepared VM images are required: one for the workers and one for the Frontend. This solution can be used only in such a Cloud where communication among the Cloud resources is manageable. This architecture also applies a single plug-in although it has to implement only the resource management and job submission functions.

4.4 Independent cloud resources with centralized job managers

The architecture and concept of the third variant is shown in Fig. 6. It can be seen that the three functions (resource management, job submission, job scheduling) of the 3G Bridge cloud plug-in are separated as independent functional units. Even more the resource management function is realized by two separate components:

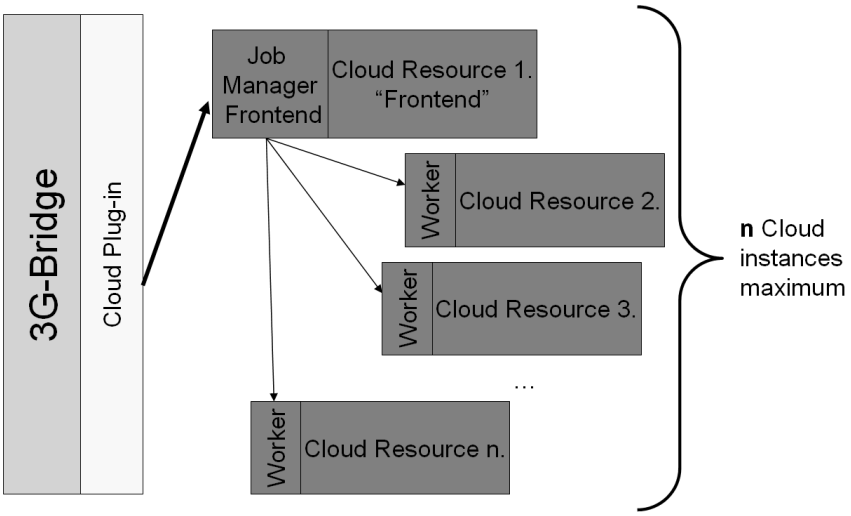


Fig. 5: Variant 2: Communicating cloud resources with centralized job manager

Cloud Plug-in and Cloud Resource Manager. The latter collects information from the 3G Bridge Condor job queue (Queue 1 in Fig. 6) about the number of waiting Condor jobs and from the cloud job queue (in our current implementation Amazon EC2 or Eucalyptus) about the available Condor worker instances. If number of worker instances has not reached yet their upper limit and there are waiting Condor jobs in the 3G Bridge Condor job queue the Cloud Resource Manager sends jobs into the Cloud Plug-in queue (Queue 2 in Fig. 6). For every such job the Cloud Plug-in will deploy a new worker instance in the cloud, and the instances will be kept running as long as the jobs in Queue 2 of the 3G-Bridge are in running state. Once an instance is deployed it will connect to the Condor Submitter/Master unit in order to get a Condor job to execute and will act as an ordinary Condor worker in a Condor pool. The instances are supplied with the address of the Condor Master at startup time, meaning different instances may connect to different masters, and thus to different pools. If there is no waiting Condor job in the 3G Bridge Condor job queue and a certain time has already spent without new arriving Condor job activity, the Cloud Resource Manager cancels some running jobs in the Cloud Plug-in queue, and thus the Cloud instances belonging to those jobs will be terminated by the Cloud plug-in.

PS-jobs arriving to 3G Bridge are submitted to the Condor Submitter/Master unit by the regular DC-API Condor Plug-in. It is the task of the Condor Submitter/Master unit to distribute the Condor jobs among the worker instances running on the cloud resources. Notice that this part of the architecture is exactly the same that is used in any 3G Bridge – Condor interconnection. This is exactly the advantage of this architecture concept that the implementation of the job submission and job scheduling functions do not require any new development.

The architecture of Fig. 6 is also very flexible. If someone would like to replace Condor with another job manager for example, PBS then it is only the DC-API Condor Plug-in that should be replaced with a PBS Plug-in. The Cloud Plug-in and the Cloud Resource manager still can be used without any modification. Similarly, if someone would like to change Amazon EC2/Eucalyptus for another type of cloud for example, OpenNebula then only the cloud interface calls of the Cloud Plug-in should be changed.

If someone would like to use several Clouds simultaneously, the architecture easily expandable to support it. In this case an additional cloud Queue and Cloud Plug-in should be added to the architecture and the Cloud Resource Manager should be extended with information collection capability from the new cloud, too.

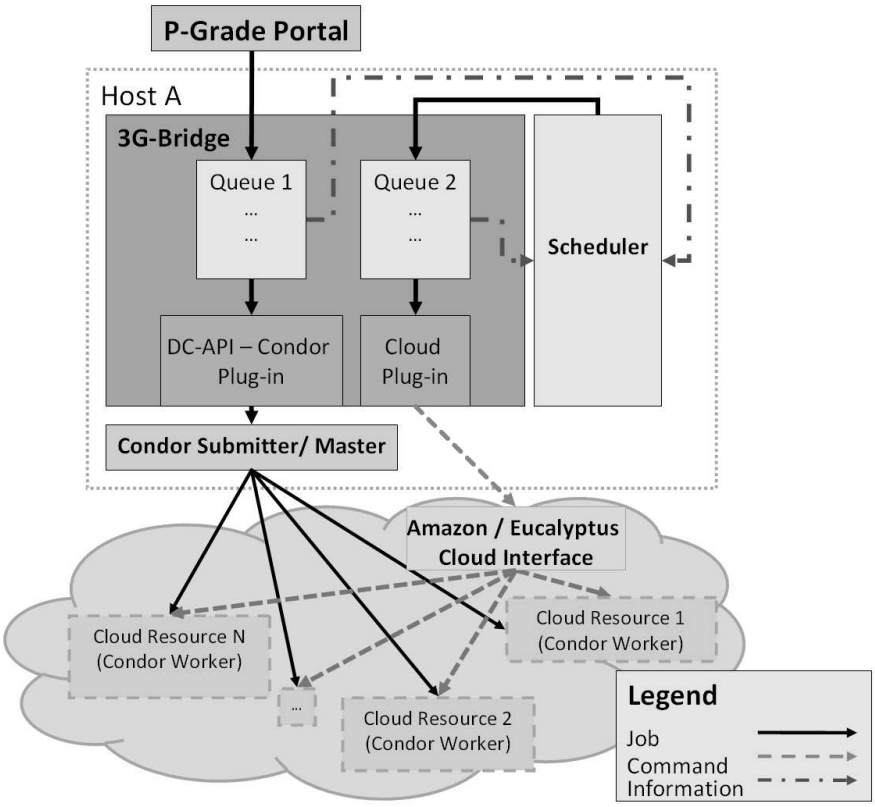


Fig. 6: Variant 3: Independent cloud resources with centralized job manager(s)

5 Performance measurements

From the three variants we choose to implement the last one (Variant 3) since it provides simplicity (only workers are running on the Cloud) with enough flexibility (Cloud resources can be started and shutdown independently, and anytime with only minor restrictions, rescheduling of work between nodes is possible). For the development we used a in-house Eucalyptus 1.51 based local Cloud, which was able to run 4 instances each with 512MB memory. For a real-world deployment we used Amazon EC2 North-American availability zone with 4 'High-CPU Medium' (c1.medium) instances to execute 80 jobs. These instances have 2 virtual cores, so 8 cores total. Each of them has increased CPU performance ('2.5 EC2 Compute Units' each), compared to the default "Small" instance ('1 EC2 Compute Units'), while only costs twice the price thus seem optimal for computation intensive tasks. Although there is a 'High-CPU Large' instance with 8 cores, that does not provide more performance per CPU or lower cost compared to the medium one, according to the information on the Amazon Web Services Website [7].

Our test was executed with the E-Marketplace Model Integrated Logistics (EMMIL) [8] application, which is solving a multi-parameter linear optimization problem to facilitate three sided negotiation between buyers, sellers and third party logistics providers. Its workflow is based on the most supported parallel-execution possibilities of P-GRADE Portal: Parameter Sweep (see Fig. 2). 3G-Bridge was used for executing the 'Parameter sweep jobs' of the EMMIL workflow. We configured EMMIL so that a single PS job would run approximately 4 minutes on a 'High-CPU Medium' Amazon EC2 instance.

For the workers we have developed our own Amazon Machine Image (AMI) similar to [9]. The image is based on Debian Linux 5.0 32-bit and Condor 7.4.2. Each instance supplied with the IP address of the Host running the 3G-Bridge and Condor thus they can automatically join the Condor pool (and leave when they are shut down). Figure 7 shows our results, namely: a.) Time-lapse of the whole execution, b.) Init phase and c.) Shutdown phase.

80 jobs were submitted to the 3G-Bridge, which then submitted the jobs to the Condor cluster. As Fig. 7/b shows 92 seconds were required for the 80 jobs to appear in Condor. Condor caused this, at certain points during the submission the 3G-Bridge started to receive "Connection refused" errors when submitting jobs from its queue to Condor. In such case the 3G-Bridge enters an exponential fallback sleep cycle between retries, with a maximum (configured) of 32 seconds. It took 159 seconds from the beginning of submission of the jobs until all 8 workers joined the pool, but a worker started the first job after 149 seconds. This is the total overhead that includes the submission to 3G-Bridge, from 3G-Bridge to Condor and the time required for the Cloud resources to join the Condor pool.

The total time for execution of the 80 tasks via 3G-Bridge on the Cloud resources took 2860 seconds. Since we did not want to submit more jobs, we issued a manual shutdown for the workers (Fig. 7/c), by issuing a cancel command for the tasks representing the running Cloud resources in the corresponding 3G-Bridge queue.

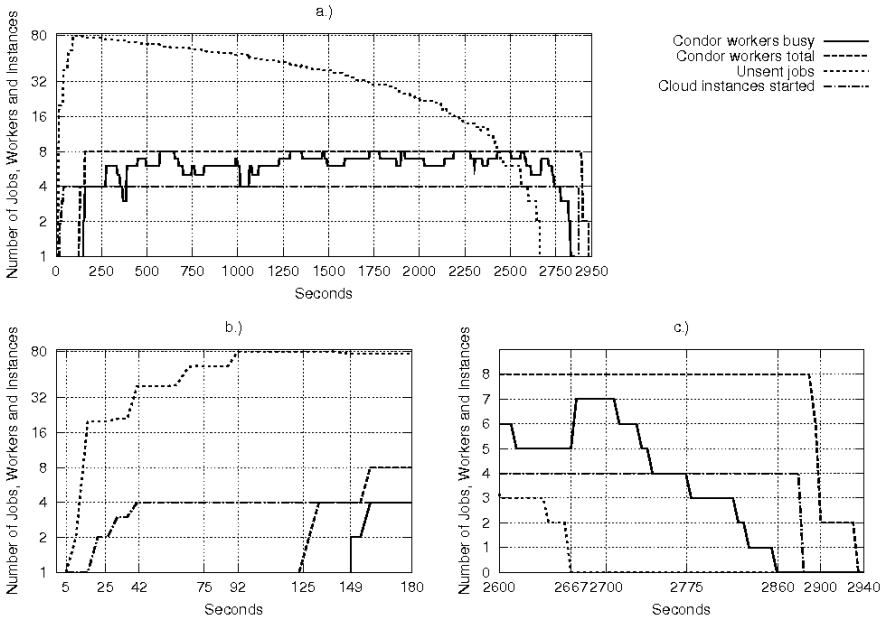


Fig. 7: EMMIL Jobs (80) executed on 4 Amazon EC2 High CPU instances via 3G-Bridge

6 Related research

Shantenu Jha et al. [10] discuss how Clouds may be treated as higher-level abstraction from Grids. For example data storage (S3 [11]) and computing (EC2) services offered by Amazon can be think of as data and storage grids with less exposed semantics (features or options) or can be considered as applications of the underlying service layer. They define the term “usage modes” for different usage patterns or deployment scenarios of applications, and “affinity” for describing the level of support of a system for a given usage mode. General-purpose Grids have large semantic feature set, since they try to address a broad variety of usage modes (and applications). They argue that as the semantic complexity decreases, the usability from the end users perspective increases. According to that, Clouds might be “closer” to end users than traditional Grids. This implies that adapting Clouds to Grid interfaces and APIs (e.g.: SAGA) first need high level abstractions for application development and deployment (direct cause of the different “affinities” of the two systems).

Chris Miceli et al. [12] were using “All-pairs” on top of SAGA (Simple API for Grid Applications) based MapReduce [13] implementation. The authors focus on data-intensive tasks, while we focus on computation-intensive tasks, but the general approach and conclusions still match. SAGA is similar to DC-API in means of providing a uniform simple API for GRID applications, but DC-API focuses on

master-worker paradigm solely on a few selected platforms rather than aiming for a generic solution. Also SAGA's runtime and (job) adaptors show similarities with the plug-in model of 3G-Bridge. The authors executed their experiments on Grids, "Cloud-like infrastructure" and Clouds (Amazon EC2, Eucalyptus and using Nimbus [14]) with up to 10 workers and with different data-set sizes ranging from 1GB to 10GB. On clouds, similar to our approach, they created a custom VM image with SAGA deployed at EC2, but used a bootstrap script that deployed SAGA and dependencies on NIMBUS and Eucalyptus.

Both approaches have their benefits, bootstrapping a "stock" VM image on each boot allows to use generic images, but an overhead is added every time the VM starts. On the other hand using custom images allows starting quickly, but requires special prepared VM images. The authors claim that EC2 took 90 seconds, while Eucalyptus took 45 seconds to instantiate a VM. It took us 120 seconds on EC2 from issuing a start command for a VM instance until the first Condor worker was ready to accept jobs, on our local Eucalyptus pool this took a lot more time, we only used Eucalyptus for development and verification. The difference of measurements on EC2 can come from many factors, like the size of the VM image used (it needs to be copied from a store every time a new instance uses it), or like the added overhead of the time required for the worker to join the Condor pool after start. Never the less, this shows that there is space for improvement there. The authors also show that there is interoperability between Grids and Clouds using SAGA: the same interface and function calls can be used whether the submission is to Grids or Clouds, this is also true for our approach, since it is only a matter of choosing the appropriate job queue at submission time.

Gideon Juve's "howto" [9] is what we used as basis for creating the Condor pool of workers in our implementation. This document gives a good starting point and general idea how to connect Condor workers running on Amazon EC2 to an arbitrary Condor pool.

Douglas Thain et al [15] describe how a Condor based cluster can act as a cloud computer and he proposes to extend the Map-Reduce abstraction for cloud computing with new abstraction layers (namely: Map, All-Pairs, Sparse-Pairs, Wavefront, Directed Graph). P-GRADE portal solution is using internally the Condor DAGMan. Condor distributed system is well-known and one of the most widely used distributed processing systems, deployed at several thousand institutions around the world, managing several hundreds thousand CPU cores in clustered environment.

Mohsen Amini et al. [16] is focusing on so called marketing-oriented scheduling policies, which can provision extra resources when the local cluster resources are not sufficient to meet the user requirements. Older scheduling policies used in Grids are not working effectively in Cloud environments, mainly because Infrastructure as a Service (IaaS) providers are charging users in a pay-as-you-go manner in an hourly basis for computational resources. To find the trade-off between to buy acquired additional resources from IaaS and reuse existing local infrastructure resources he proposes two scheduling policies (Cost and Time Optimization scheduling policies) for mixed (commercial and non-commercial) resource environments. The evaluation of the policies was done in Gridbus broker that is originally able to interface

with Condor and SGE. However, he is also extending the broker to interact with Amazon EC2. Basically two different approaches were identified on provisioning commercial resources. The first approach is offered by the IaaS providers at resource provisioning level (user/application constraints are neglected: deadline, budget, etc.), the other approach deploys resources focusing at user level (time and/or cost minimization, estimating the workload in advance, etc.).

OpenNebula [17] can hire commercial (Amazon EC2) resources in an on-demand manner, when the local cluster resources are overloaded. Llorente et al. [18] extend OpenNebula to provision additional resources for applications to handle peak load, when the local cluster or grid environment is saturated. Silva et al. [19] proposes a mechanism for creating and deploying optimal amount of resources in Cloud environment, when the workload is hard to predict due to the usage of Bag-of-Task applications.

One solution of application deployment on Cloud infrastructure is detailed in [20], which introduce a way to distribute a parameter sweep type application with Aneka (Cloud development and management platform) on Cloud resources. In Aneka enterprise clouds cost-optimization, time-optimization, and conservative time-optimization scheduling algorithms are available.

7 Conclusion

This chapter introduced the existing connectivity and interoperability issues of Clouds, Grids and Clusters, and showed solutions how such issues can be solved. Firstly we have enumerated the main pillars of e-science infrastructure eco-systems which are recently extended by Cloud infrastructure. Then the basic principles of parameter sweep job concept and its execution by P-GRADE portal was briefly explained. We have then examined three solution variants in details for processing parameter sweep jobs on Cloud resources. From these three alternatives, the last variant – the most flexible solution using the 3G-Bridge developed by the EDGeS project to interconnect different (grid) middleware – was implemented. With performance measurements we have evaluated the implemented solution. Our results show clearly, that this solution allows use on-demand cloud resources in a transparent and efficient way with improved scalability. Future work includes performing larger scale tests and integrating this solution into the EDGeS infrastructure. We also need to handle different requirements of the tasks (e.g.: currently we only support 32 bit Linux environment, and no extra dependencies or requirements may be defined for the tasks).

References

1. E. Urbah, P. Kacsuk, Z. Farkas, G. Fedak, G. Kecskemeti, O. Lodygensky, A. Marosi, Z. Balaton, G. Caillat, G. Gombas, A. Kornafeld, J. Kovacs, H. He, and R. Lovas. EDGeS: Bridging EGEE to BOINC and XtremWeb, Journal of Grid Computing, Vol. 7, No. 3, pp. 335-354, 2009
2. P. Kacsuk and G. Sipos: Multi-Grid, Multi-User Workflows in the P-GRADE Portal, Journal of Grid Computing, Vol. 3, No. 3-4, Springer, pp. 221-238, 2005.
3. P. Kacsuk, T. Kiss, G. Sipos: Solving the grid interoperability problem by P-GRADE portal at workflow level, Future Generation Computer Systems, Volume 24, Issue 7, July 2008, Pages 744-751, 2008
4. P. Kacsuk, Z. Farkas; G. Herman. Workflow-level parameter study support for production grids. Computational science and its application - ICCSA 2007. International conference. Part III. Kuala Lumpur, 2007. (Lecture notes in computer science 4707.),: 2007.: Berlin, ISBN: 978-3-540-74482-5, pp. 872-885
5. Attila Csaba Marosi, Gábor Gombás, Zoltán Balaton, Péter Kacsuk: Enabling Java applications for BOINC with DC-API. In Distributed and Parallel Systems, proceedings of the 7th International Conference On Distributed And Parallel Systems, pp3-12, 2009
6. P. Kacsuk, J. Kovács; Z. Farkas; A. Marosi; G. Gombás; Z. Balaton: SZTAKI desktop grid (SZDG): a flexible and scalable desktop grid system., 2009. Journal of Grid Computing Vol.: 7 , No. 4, pp. 439-461
7. Amazon EC2 Instance Types: `http://aws.amazon.com/ec2/instance-types/`, Cited 13 May 2010
8. Kacsuk ,P, Kacsukné B. L., Hermann, G., Balasko, A (2007): Simulation of EMMIL E-Marketplace Model in the P-GRADE Grid Portal, Proceedings of ESM'2007 International Conference, Malta, pp. 569-573.
9. Condor Workers on Amazon EC2: `http://www-rcf.usc.edu/~juve/condor-ec2/`, Cited 13 May 2010
10. Jha, Shantenu and Merzky, Andre and Fox, Geoffrey, Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes, Concurr. Comput. : Pract. Exper.Vol.21Num. :8,2009, issn :1532-0626, pp.1087 – 1108, John Wiley and Sons Ltd., Chichester, UK
11. Amazon Simple Storage Service (S3) `https://aws.amazon.com/s3`, Cited 13 May 2010
12. Chris Miceli and Michael Miceli and Shantenu Jha and Hartmut Kaiser and Andre Merzky, Programming Abstractions for Data Intensive Computing on Clouds and Grids, Cluster Computing and the Grid, IEEE International Symposium on,Vol.:0,ISBN 978-0-7695-3622-4, 009, pp.:478-483, IEEE Computer Society, Los Alamitos, CA, USA
13. Dean, Jeffrey and Ghemawat, Sanjay,MapReduce: simplified data processing on large clusters, Commun. ACM,Vol.:51, Num:1,2008, ISSN :0001-0782, pp.:107 – 113, ACM, New York, NY, USA
14. NIMBUS. `http://www.nimbusproject.org/`, Cited 13 May 2010
15. Douglas Thain and Christopher Moretti. Abstractions for Cloud Computing with Condor, Syed Ahson and Mohammad Ilyas, Cloud Computing and Software Services, CRC Press, December, 2009. ISBN: 9781439803158
16. Mohsen Amini Salehi and Rajkumar Buyya, Adapting Market-Oriented Scheduling Policies for Cloud Computing, 10th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP 2010, May 21-23, 2010, Busan, Korea
17. J. Fontán, T. Vázquez, L. Gonzalez, RS Montero, and IM Llorente. OpenNEbula: The open source virtual machine manager for cluster computing. In Open Source Grid and Cluster Software Conference, 2008.
18. I. Llorente, R. Moreno-Vozmediano, and R. Montero. Cloud computing for on-demand grid resource provisioning. Advances in Parallel Computing, 2009.

19. J.N. Silva, L. Veiga, and P. Ferreira. Heuristic for resources allocation on utility computing infrastructures. In Proceedings of the 6th international workshop on Middleware for grid computing, pp. 9-17, 2008.
20. Cloud Based Parameter Sweeping of a Repast Simphony Simulation Model-GUI Mode,Technical Note,24/02/2010, http://community.decisci.com/content/cloud-based-parameter-sweeping-repast-\-simphony-simulation-model-gui-mode#content , Cited 13 May 2010

Energy Aware Clouds

Anne-Cécile Orgerie, Marcos Dias de Assunção and Laurent Lefèvre

Abstract Cloud infrastructures are increasingly becoming essential components for providing Internet services. By benefitting from economies of scale, Clouds can efficiently manage and offer a virtually unlimited number of resources, and can minimise the costs incurred by organisations when providing Internet services. However, as Cloud providers often rely on large data centers to sustain their business and offer the resources that users need, the energy consumed by Cloud infrastructures has become a key environmental and economical concern. This chapter presents an overview of techniques that can improve the energy efficiency of Cloud infrastructures. We propose a framework termed as Green Open Cloud, which uses energy efficient solutions for virtualised environments; the framework is validated on a reference scenario.

1 Introduction

Cloud solutions have become essential to current and future Internet architectures as they provide on-demand virtually unlimited numbers of compute, storage and network resources. This elastic characteristic of Clouds allows for the creation of computing environments that scale up and down according to the requirements of distributed applications.

Anne-Cécile Orgerie
ENS Lyon - LIP Laboratory (UMR CNRS, INRIA, ENS, UCB), University of Lyon - 46 allée d'Italie 69364 Lyon Cedex 07 - France, e-mail: annececile.orgerie@ens-lyon.fr

Marcos Dias de Assunção
INRIA - LIP Laboratory (UMR CNRS, INRIA, ENS, UCB) - University of Lyon - 46 allée d'Italie 69364 Lyon Cedex 07 - France, e-mail: marcos.dias.de.assuncao@ens-lyon.fr

Laurent Lefèvre
INRIA - LIP laboratory (UMR CNRS, INRIA, ENS, UCB) - University of Lyon - 46 allée d'Italie 69364 Lyon Cedex 07 - France, e-mail: laurent.lefevre@inria.fr

Through economies of scale, Clouds can efficiently manage large sets of resources; a factor that can minimise the cost incurred by organisations when providing Internet services. However, as Cloud providers often rely on large data centers to sustain their business and supply users with the resources they need, the energy consumed by Cloud infrastructures has become a key environmental and economical concern. Data centers built to support the Cloud computing model can usually rely on environmentally unfriendly sources of energy such as fossil fuels [2].

Clouds can be made more energy efficient through techniques such as resource virtualisation and workload consolidation. After providing an overview of energy-aware solutions for Clouds, this chapter presents an analysis of the cost of virtualisation solutions in terms of energy consumption. It also explores the benefits and drawbacks that Clouds could face by deploying advanced functionalities such as Virtual Machine (VM) live migration, CPU throttling and virtual CPU pinning. This analysis is important for users and administrators who want to devise resource allocation schemes that endeavour to reduce the CO_2 footprint of Cloud infrastructures.

As we attempt to use recent technologies (*i.e.* VM live migration, CPU capping and pinning) to improve the energy efficiency of Clouds, our work can be positioned in the context of future Clouds. This work proposes an energy-aware framework termed as Green Open Cloud (GOC) [21] to manage Cloud resources. Benefiting from the workload consolidation [34] enabled by resource virtualisation, the goal of this framework is to curb the energy consumption of Clouds without sacrificing the quality of service (in terms of performance, responsiveness and availability) of user applications. All components of the GOC architecture are presented and discussed: Green policies, prediction solutions and network presence support. We demonstrate that under a typical virtualised scenario GOC can reduce the energy used by Clouds by up to 25% compared to basic Cloud resource management.

The remaining part of this chapter is organised as follows. Section 2 discusses energy-aware solutions that can be applied to Clouds. Then, the energy cost of virtual machines is investigated in Section 3. The GOC architecture is described in Section 4 and evaluated on a typical virtualised scenario Section 5.

2 Overview of Energy Aware Techniques for Clouds

Current Web applications demand highly flexible hosting and resource provisioning solutions [35]. The rising popularity of social network Web sites, and the desire of current Internet users to store and share increasing amounts of information (*e.g.* pictures, movies, life-stories, virtual farms) have required scalable infrastructure. Benefiting from economies of scale and recent developments in Web technologies, data centers have emerged as a key model to provision resources to Web applications and deal with their availability and performance requirements. However, data centers are often provisioned to handle sporadic peak loads, which can result in low resource utilisation [15] and wastage of energy [12].

The ever-increasing demand for cloud-based services does raise the alarming concern of data center energy consumption. Recent reports [29] indicate that energy costs are becoming dominant in the Total Cost of Ownership (TCO). In 2006, data centers represented 1.5 percent of the total US electricity consumption. By 2011, the current data center energy consumption could double [31] leading to more carbon emissions. Electricity becomes the new limiting factor for deploying data center equipments.

A range of technologies can be utilised to make cloud computing infrastructures more energy efficient, including better cooling technologies, temperature-aware scheduling [24, 9, 30], Dynamic Voltage and Frequency Scaling (DVFS) [14, 33], and resource virtualisation [36]. The use of VMs [3] brings several benefits including environment and performance isolations; improved resource utilisation by enabling workload consolidation; and resource provisioning on demand. Nevertheless, such technologies should be analysed and used carefully for really improving the energy-efficiency of computing infrastructures [23]. Consolidation algorithms have to deal with the relationship between performance, resource utilisation and energy, and can take advantage from resource heterogeneity and application affinities [34]. Additionally, techniques such as VM live-migration [6, 13], can greatly improve the capacities of Cloud environments by facilitating fault management, load balancing and lowering system maintenance costs. VM migration provides a more flexible and adaptable resource management and offers a new stage of virtualisation by removing the concept of locality in virtualised environments [38].

The overhead posed by VM technologies has decreased over the years, which has expanded their appeal for running high performance computing applications [37] and turned virtualisation into a mainstream technology for managing and providing resources for a wide user community with heterogeneous software-stack requirements. VM-based resource management systems such as Eucalyptus [26] and OpenNebula [10], allow users to instantiate and customise clusters of virtual machines atop the underlying hardware infrastructure. When applied in a data center environment, virtualisation can allow for impressive workload consolidation. For instance, as Web applications usually present variable user population and time-variant workloads, virtualisation can be employed to reduce the energy consumed by the data center environment through server consolidation whereby VMs running different workloads can share the same physical host. By consolidating the workload of user applications into fewer machines, unused servers can potentially be switched off or put in low energy consumption modes. Yet attracting virtualisation is, its sole use does not guarantee reductions in energy consumption. Improving the energy efficiency of Cloud environments with the aid of virtualisation generally calls for devising mechanisms that adaptively provision applications with resources that match their workload demands and utilises other power management technologies such as CPU throttling and dynamic reconfiguration; allowing unused resources to be freed or switched off.

Existing work has proposed architectures that benefit from virtualisation for making data centers and Clouds more energy efficient. The problem of energy-efficient resource provisioning is commonly divided into two subproblems [22]: at micro-

or host level, power management techniques are applied to minimise the number of resources used by applications and hence reduce the energy consumed by an individual host; and at a macro-level, generally a Resource Management System (RMS) strives to enforce scheduling and workload consolidation policies that attempt to reduce the number of nodes required to handle the workloads of user applications or place applications in areas of a data center that would improve the effectiveness of the cooling system. Some of the techniques and information commonly investigated and applied at a macro- or RMS-level to achieve workload consolidation and energy-efficient scheduling include:

- Applications workload estimation;
- The cost of adaptation actions;
- Relocation and live-migration of virtual machines;
- Information about server-racks, their configurations, energy consumption and thermal states;
- Heat management or temperature-aware workload placement aiming for heat distribution and cooling efficiency;
- Study of application dependencies and creation of performance models; and
- Load balancing amongst computing sites;

Server consolidation has been investigated in previous work [8, 39, 40, 5, 18, 20, 16, 34]. A key component of these systems is the ability to monitor and estimate the workload of applications or the arrival of user requests. Several techniques have been applied to estimate the load of a system, such as exponential moving averages [4], Kalman filters [17], autoregressive models, and combinations of methods [19, 5]. Provisioning VMs in an IaaS environment poses additional challenges as information about the user applications is not always readily available. Section 4.3 describes an algorithm for predicting the characteristics of advance reservation requests that resemble requests for allocating virtual machines.

Fitted with workload-estimation techniques, these systems provide schemes to minimise the energy consumed by the underlying infrastructure while minimising costs and violations of Service Level Agreements (SLAs). Chase *et al.* [5] introduced MUSE, an economy-based system that allocates resources of hosting centers to services aiming to minimise energy consumption. Services bid for resources as a function of delivered performance whilst MUSE switches unused servers off. Kalyvianaki *et al.* [18] introduced autonomic resource provisioning using Kalman filters. Kusic *et al.* proposed a lookahead control scheme for constantly optimising the power efficiency of a virtualised environment [20]. With the goal of maximising the profit yielded by the system while minimising the power consumption and SLA violations, the provisioning problem is modelled as a sequential optimisation under uncertainty and is solved using the lookahead control scheme. Placement of applications and scheduling can also take into account the thermal states or the heat

dissipation in a data center [24]. The goal is scheduling workloads in a data center, and the heat they generate, in a manner that minimises the energy required by the cooling infrastructure, hence aiming to minimise costs and increase the overall reliability of the platform.

Although consolidation fitted with load forecasting schemes can reduce the overall number of resources used to serve user applications, the actions performed by RMSs to adapt the environment to match the application demands can require the relocation and reconfiguration of VMs. That can impact the response time of applications, consequently degrading the QoS perceived by end users. Hence, it is important to consider the costs and benefits of the adaptation actions [40]. For example, Gueyoung *et al.* [16] have explored a cost-sensitive adaptation engine that weights the potential benefits of reconfiguration and their costs. A cost model for each application is built offline and to decide when and how to reconfigure the VMs, the adaptation engine estimates the cost of adaptation actions in terms of changes in the utility, which is a function of the application response time. The benefit of an action is given by the improvement in application response time and the period over which the system remains in the new configuration.

Moreover, consolidation raises the issue of dealing with necessary redundancy and placement geo-diversity at the same time. Cloud providers, as Salesforce.com for example, that offer to host entire websites of private companies [11], do not want to lose entire company websites because of power outages or network access failures. Hence, outage and blackout situations should be anticipated and taken into account in the resource management policies [32].

While the macro-level resource management performs actions that generally take into account the power consumption of a group of resources or the whole data center, at the host-level the power management is performed by configuring parameters of the hypervisor's scheduler, such as throttling of Virtual CPUs (VCPU) and using other OS specific policies. In the proposed architectures, hosts generally run a local resource manager that is responsible for monitoring the power consumption of the host and optimising it according to local policies. The power management capabilities available in virtualised hosts has been categorised as [25]: "soft" actions such as CPU idling and throttling; "hard" actions like DVFS; and consolidating in the hypervisor. CPU idling or soft states consist in changing resource allotments of VMs and attributes of the hypervisor's scheduler (*e.g.* number of credits in Xen's credit scheduler) to reduce the CPU time allocated to a VM so that it consumes less power. Hard actions comprise techniques such as scaling the voltage and frequency of CPUs. Consolidation can also be performed at the host-level where the VCPUs allocated to VMs can be configured to share CPU cores, putting unused cores in idle state, hence saving the energy that would otherwise be used by the additional core to run a VM.

Nathuji and Schwan [25] presented VirtualPower, a power management system for virtualised environments that explores both hardware power scaling and software-based methods to control the power consumption of underlying platforms. VirtualPower exports a set of power states to VM guests that allow guests to use and act upon these states thus performing their own power management policies.

The soft states are intercepted by Xen hypervisor and are mapped to changes in the underlying hardware such as CPU frequency scaling according to the virtual power management rules. The power management policies implemented in the guest VMs are used as "hints" by the hypervisor rather than executable commands. They also evaluate the power drawn by cores at different frequency/voltage levels and suggest that such technique be used along with soft schemes.

3 Investigating the Energy Consumption of Virtual Machines

With the aim of integrating soft schemes such as CPU throttling, the next sections describe simple experiments to evaluate the distance between the idle consumption and the consumption at high utilisation of virtualised servers. The experiments evaluate the additional power drawn by VMs and the impact of operations such as CPU idling, consolidation at the host level and VM live-migration.

3.1 Experimental Scenario

The evaluation has been performed on a testbed composed of HP Proliant 85 G2 servers (2.2GHz, 2 duo core CPUs per node) with Xen open source 3.4.1. Each node is connected to an external wattmeter that logs its instant power consumption; one measurement is taken each second. The storage of these energy logs is performed by a data collector machine. The precision of measurements of this setup is 0.125 Watts whereas the frequency of measurements is one second.

3.2 Virtual Machine Cost

In order to be energy efficient, virtual machines should ideally start and halt quickly and without incurring too much energy usage. It is hence crucial to understand the cost of basic virtual machine operations and statuses (such as boot, run, idle, halt) in terms of energy consumption.

The experiments reported here describe the stages of booting, running and shutting down virtual machines. Each virtual machine is configured to use one VCPU without pinning (*i.e.* we do not restrict which CPUs a particular VCPU may run on using the generic *vcpu-pin* interface), and no capping of CPU credits. We measure the energy consumption by running different configurations, one at a time, each with a different number of virtual machines; from one to six virtual machines on one physical resource.

The graph in Figure 1 shows the energy consumption of virtual machines that are initialised, but do not have a workload (*i.e.*, they do not execute any application after

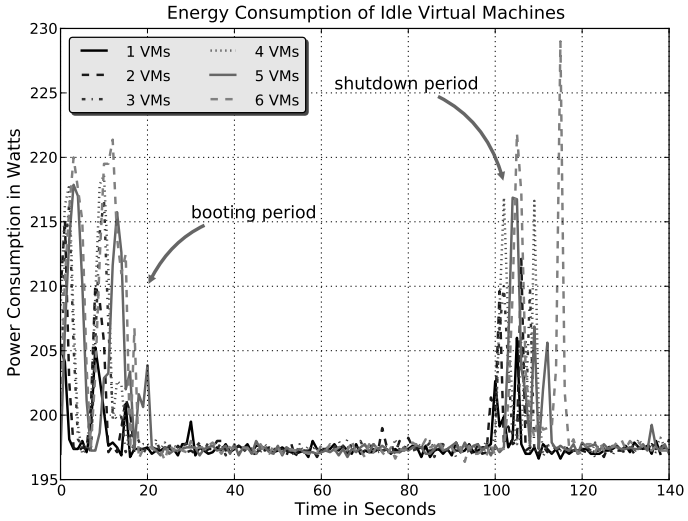


Fig. 1: Energy consumption of different numbers of idle virtual machines on one physical machine.

booting). As shown by this figure, the start and shutdown phases of VMs consume an amount of energy that should not be ignored when designing resource provisioning or consolidation mechanisms. The graph in Figure 2, on the other hand, shows the consumption of virtual machines that execute a CPU intensive sample application (*i.e.* cpuburn[1]) once they finish booting. The CPU intensive workload runs for 60 seconds once the virtual machine is initialised and we wait for some time before shutting it down to recognise more clearly the shutdown stage in the graphs.

As shown by Figure 2, the increase in energy consumption resulting from increasing the number of virtual machines in the system depends largely on the number of cores in use. Although the 5th and 6th VMs seem to come at a zero cost in terms of energy consumption since all the cores were already in use, in reality the application performance can be degraded. We have not evaluated the performance in this work since we run a sample application, but in future we intend to explore the performance degradation and the trade-off between application performance and energy savings.

3.3 Migration Cost

This experiment evaluates the energy consumption when performing live migration of four virtual machines running a CPU intensive workload. After all virtual

[1] cpuburn is a test application that attempts to use 100% of CPU.

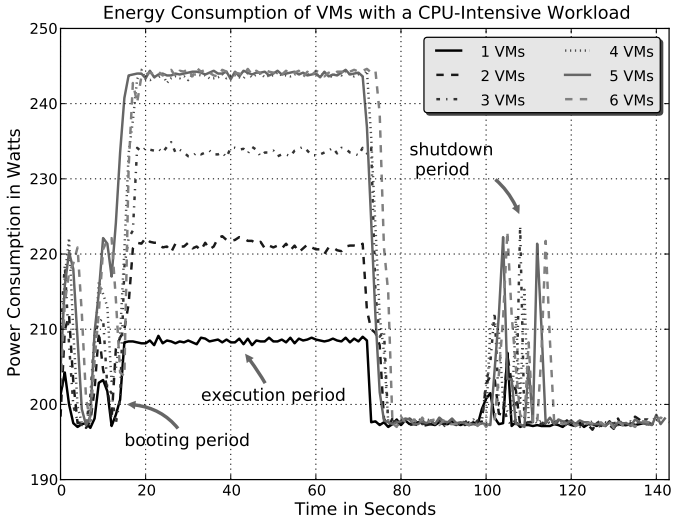


Fig. 2: Energy consumption of virtual machines running a CPU intensive workload on a shared physical resource.

machines are initialised, at time 40 seconds, the live migration starts; one virtual machine is migrated at every 30 seconds. The results are summarised in Figure 3.

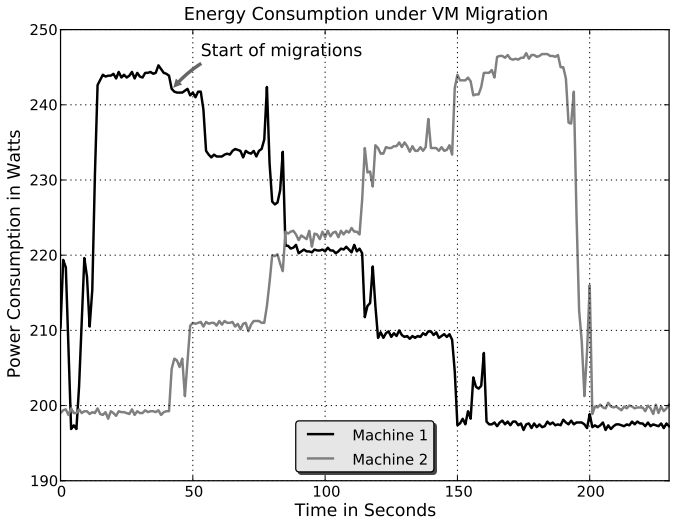


Fig. 3: Migration of 4 virtual machines starting after 40 seconds, one migration every 30 seconds.

The figure shows that migrating VMs leads to an increase in energy consumption during the migration time – factor evidenced by the asymmetry between the lines. Even though the VMs migrated in this scenario had only 512MB of RAM, the experiment demonstrates that when migrating with the goal of shutting down unused resources, one must consider that during the migration, two machines will be consuming energy.

3.4 Capping and Pinning of VCPUs

This experiment measures the energy consumption of virtual machines when we vary parameters of the credit scheduler used by Xen hypervisor. The first parameter evaluated is the cap of CPU utilisation. Valid values for the cap parameter when using one virtual CPU are between 1 and 100, and the parameter is set using the *xm sched-credit* command. Initially, we run a virtual machine with a CPU intensive workload and measure the energy consumption as we change the CPU cap. The experiment considers the following caps: 100, 80, 60, 40 and 20. Once the virtual machine is initialised and the CPU intensive workload is started, the virtual machine remains during one minute under each cap and is later shut down. The results are summarised in Figure 4.

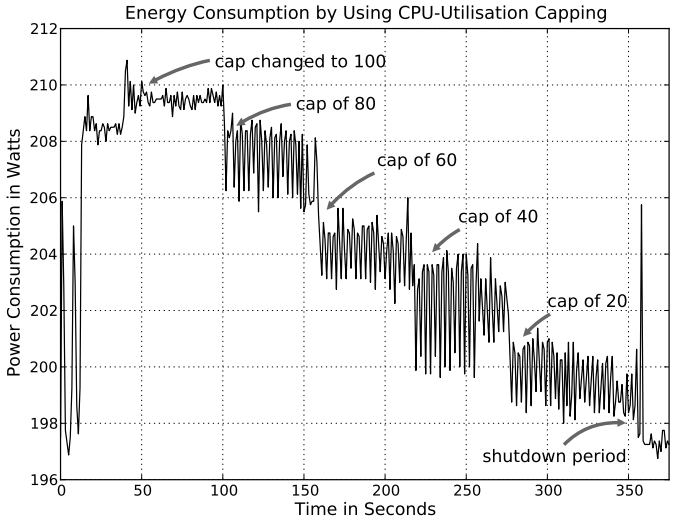


Fig. 4: Energy consumption of a virtual machine under different CPU utilisation caps.

Although capping has an impact on power consumption, this impact is small when considering only one core, and under some cap values the energy consumption does not remain stable. However, the difference in consumption is more noticeable when throttling the CPU usage of multiple VMs.

When virtual machines are initialised, the credit scheduler provided by Xen hypervisor is responsible for deciding which physical cores the virtual CPUs will utilise. However, the assignment of virtual CPUs to physical cores can be modified by the administrator via Xen's API by restricting the physical cores used by a virtual machine; operation known as "pinning". The second set of experiments described here evaluate the energy consumption when changing the default pinning carried out by Xen's hypervisor.

This experiment first initialises two VMs with the CPU intensive workload; then, leaves them run under default pinning for one minute; after that, throttles the VCPUs by forcing them to share the same core; next, changes the core to which the VCPUs are pinned at each minute; after that, removes pin restrictions; and finally shuts down the VMs. Figure 5 summarises the results.

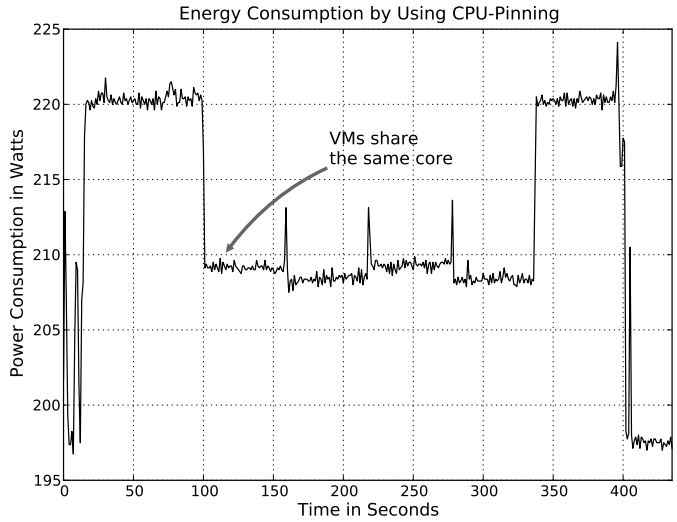


Fig. 5: Energy consumption by making the virtual CPUs of two VMs share the same core.

The energy consumption reduces as the number of utilised cores in the system decreases. Although this may look obvious, the results demonstrate that it is possible to consolidate a workload at the host level and minimise the energy consumption by using capping. For example, a hosting center may opt for consolidating workloads if a power budget has been reached or to minimise the heat produced by a group of servers. Our future work will investigate the trade-off's between performance

degradation and energy savings achieved by VCPU throttling when provisioning resources to multi-tier Web applications.

4 The Green Open Cloud

4.1 The Green Open Cloud Architecture

The energy footprint of current data centers has become a critical issue, and it has been shown that servers consume a great amount of electrical power even when they are idle [28]. Existing Cloud architectures do not take full advantage of recent techniques for power management, such as Virtual Machine (VM) live migration, advance reservations, CPU idling, and CPU throttling [25].

We attempt to make use of recent technologies to improve the energy efficiency of Clouds, thus placing our work in the context of future Clouds. This work proposes an energy-aware framework termed as Green Open Cloud (GOC) [21] to manage Cloud resources. Benefitting from the workload consolidation [34] enabled by resource virtualisation, the goal of this framework is to curb the energy consumption of Clouds without sacrificing the quality of service (in terms of performance, responsiveness and availability) of user applications.

In the proposed architecture, users submit their reservation requests via a Cloud portal (*e.g.* a Web server) (Figure 6). A reservation request submitted by a user to the portal contains the required number of VMs, its duration and the wished start time. GOC manages an agenda with resource reservations; all reservations are strict, and once approved by GOC, their start times cannot change. This is like a green Service Level Agreement (SLA) where the Cloud provider commits to give access to required resources (VMs) during the entire period that is booked in the agenda. This planning provides a great flexibility to the provider, that can have a better control on how the resources are provisioned.

Following the pay-as-you-go philosophy of Clouds, the GOC framework delivers resources in a pay-as-you-use manner to the provider: only utilised resources are powered on and consume electricity. The first step to reduce electricity wastage is to shut down physical nodes during idle periods. However, this approach is not trivial as a simple on/off policy can be more energy consuming than doing nothing because powering nodes off and on again consumes electricity and takes time. Hence, GOC uses prediction algorithms to avoid frequent on/off cycles. VM migration is used to consolidate the load into fewer resources, thus allowing the remaining resources to be switch off. VCPU pinning and capping are used as well for workload consolidation. GOC can also consolidate workloads considering time by aggregating resource reservations. When a user submits a reservation request, the Green Open Cloud framework suggests alternative start times for the reservation request along with the predicted amount of energy it will consume under the different scenarios. In this way, the user can make an informed choice and favour workload aggregation in

time, hence avoiding excessive on/off cycles. On/off algorithms also pose problems for resource management systems, which can interpret a switched-off node as a failure. To address this issue, GOC uses a trusted proxy that ensures the nodes' network presence; the Cloud RMS communicates with this proxy instead of the switched-off nodes (Figure 6).

The key functionalities of the GOC framework are to:

- monitor Cloud resources with energy sensors to take efficient management decisions;
- provide energy usage information to users;
- switch off unused resources to save energy;
- use a proxy to ensure network presence of switched-off resources and thus provide inter-operability with different RMSs;
- use VM migration to consolidate the load of applications in fewer resources;
- predict the resource usage to ensure responsiveness; and
- give "green" advice to users in order to aggregate resource requests.

The GOC framework works as an overlay atop existing Cloud resource managers. GOC can be used with all types of resource managers without impacting on their workings, such as their scheduling policies. This modular architecture allows a great flexibility and adaptivity to any type of future Cloud's RMS architecture. The GOC framework relies on energy sensors (wattmeters) to monitor the electricity consumed by the Cloud resources. These sensors provide direct and accurate assessment of GOC policies, which helps it in using the appropriate power management solutions.

The GOC architecture, as described in Figure 6, comprises:

- a set of energy sensors providing dynamic and precise measurements of power consumption;
- an energy data collector which stores and provides the energy logs through the Cloud Web portal (to increase the energy-awareness of users);
- a trusted proxy for supporting the network presence of switched-off Cloud resources; and
- an energy-aware resource manager and scheduler which applies the green policies and gives green advice to users.

The Cloud RMS manages the requests in coordination with the energy-aware resource manager which is permanently linked to the energy sensors. The energy-aware resource manager of GOC can be implemented either as an overlay on the existing Cloud RMS, or as a separate service. Figure 7 depicts a scenario where GOC is implemented as an overlay on the existing Cloud RMS. In the resource manager, the white boxes represent the usual components of a future Cloud RMS (with agenda) whereas the shaded boxes depict the GOC functionalities. These add-

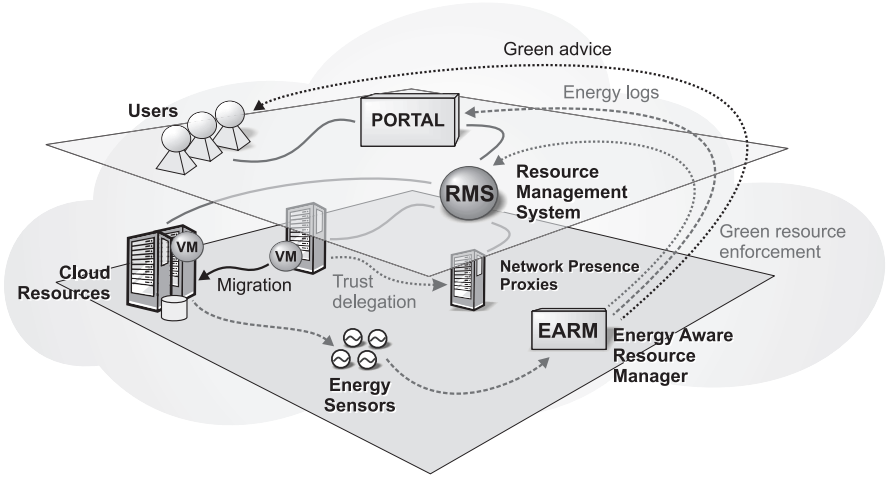


Fig. 6: The GOC architecture.

ons are connected to the RMS modules and have access to the data provided by users (*i.e.* submitted requests) and the data in the reservation agenda.

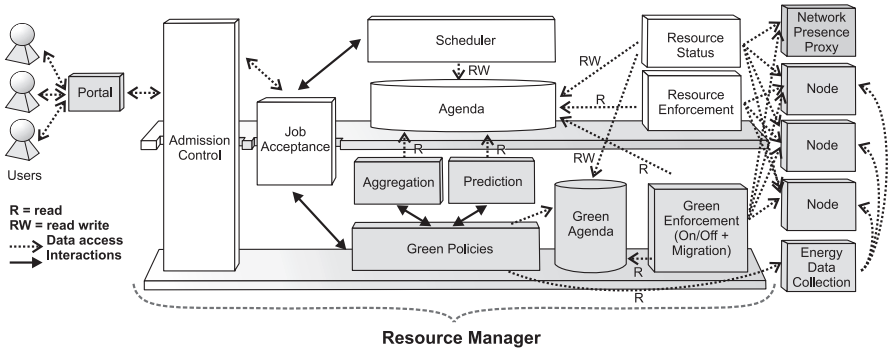


Fig. 7: The GOC resource manager.

When a user submits a request, the *admission control* module checks whether it can be admitted into the system by attempting to answer questions such as: Is the user allowed to use this Cloud? Is the request valid? Is the request compliant with the Cloud's usage chart? Then, the *job acceptance* module transfers the request to the *scheduler* and to the *green resource manager* (also called energy-aware manager and labeled "*Green Policies*" in Figure 7). The *scheduler* queries the *agenda* to see whether the requested reservation can be placed into it whilst the *green resource manager* uses its *aggregation* and *prediction* modules to find other less energy-consuming possibilities to schedule this reservation in accordance with its green

policies. All the possible schedules (from the Cloud's *scheduler* and from the *green resource manager*) are then transferred to the *job acceptance* module which presents them to the user and prompts her for a reply.

The *agenda* is a database containing all the future reservations and the recent history (used by the prediction module). The *green agenda* contains all the decisions of the green resource manager: on/off and VM migrations (when to switch on and off the nodes and when to migrate VMs). These decisions are applied by the *green enforcement* module. The *resource enforcement* module manages the reservations (gives the VMs to the users) in accordance with the *agenda*. The *resource status* component periodically polls the nodes to know whether they have hardware failures. If the nodes are off, the component queries the *network presence proxy*, so the nodes are not woken up unnecessarily. The *energy data collector* module monitors the energy usage of the nodes and gives access to this information to the *green resource manager*. These data are also put on the Cloud Web portal, so users can see their energy impact on the nodes and increase their energy-awareness.

4.2 Network Presence

As we switch off unused nodes, they do not reply to queries made by the Cloud resource manager, which can be considered by the RMS as a resource failure. This problem is solved by using a trusted proxy to ensure the network presence of the Cloud resources. When a Cloud node is switched off, all of its basic services (such as ping or heartbeat services) are migrated to the proxy that will then answer on behalf of the node when asked by the resource manager. The key issue is to ensure the security of the infrastructure and to avoid the intrusion of malicious nodes. Our trust-delegation model is described in more details in previous work [7]. This mechanism allows a great adaptivity of the GOC framework to any Cloud RMS.

4.3 Prediction Algorithms

As reservations finish, the prediction algorithm predicts the next reservation for the freed resources. If the predicted reservation of a resource is too close (in less than T_s seconds), the resource remains powered on (it would consume more energy to switch it off and on again), otherwise it is switched off. This idea is illustrated by Figure 8.

For a given resource, T_s is given by the following formula:

$$T_s = \frac{E_{ON \to OFF} + E_{OFF \to ON} - P_{OFF}(\delta_{ON \to OFF} + \delta_{OFF \to ON})}{P_{idle} - P_{OFF}}$$

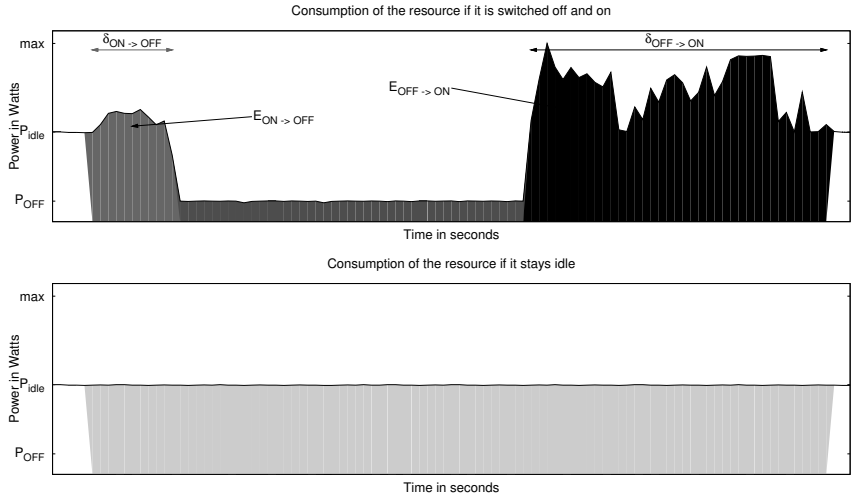


Fig. 8: Definition of T_s

where P_{idle} is the idle consumption of the resource (in Watts), P_{OFF} the power consumption when the resource is off (in Watts), $\delta_{ON \rightarrow OFF}$ the duration of the resource shutdown (in seconds), $\delta_{OFF \rightarrow ON}$ the duration of the resource boot, $E_{ON \rightarrow OFF}$ the energy consumed to switch off the resource (in Joules) and $E_{OFF \rightarrow ON}$ the energy consumed to switch on the resource (in Joules).

In the same way, we define temporal parameters based on the energy consumption of the resources to know in which case it would be more energy efficient to use VM migration. T_m is the bound on the remaining reservation's duration beyond which migration is more energy efficient: if the reservation is still running for more than $T - m$ seconds, the VMs should be migrated, otherwise they stay on their respective physical hosts. As the migration takes time, migration of small jobs is not useful (with a duration lower than 20 seconds in that example). This lower time bound is denoted T_a. If a migration is allowed, the *green enforcement* module also waits T_a seconds after the beginning of a job before migrating it. It indeed allows initialising the job and the VM, so the migration is simplified and we avoid the migration of small jobs, crashed jobs or VMs (in case of technical failure or bad configuration of the VM).

For the next reservation, the predicted arrival time is the average of the inter-submission time of the previous jobs plus a feedback. The feedback is computed with the previous predictions; it represents an average of the errors made by computing the n previous predictions (n is an integer). The error is the difference between the true observation and the predicted value. For estimating other features of the reservation (size in number of resources and length), the same kind of algorithm is used (average of the previous values at a given time).

We have seen in previous work [27] that even with a small n (5 for example) we can obtain good results (70% of good predictions on experimental Grid traces). This prediction model is simple, but it does not need many accesses and is really fast to compute, which are crucial features for real-time infrastructures. This prediction algorithm has several advantages: it works well, it requires a small part of history (no need to store big amounts of data) and it is fast to compute. Therefore, it will not delay the whole reservation process, and as a small part of the history is used, it is really responsive and well adapted to request bursts as it often occurs in such environments.

4.4 Green Policies

Among the components of a Cloud architecture, up to now, we have focused on virtualisation, which appears as the main technology used in these architectures. It also uses migration to dynamically unbalance the load between the Cloud nodes in order to shut down some nodes, and thus save energy.

However, GOC algorithms can employ other Cloud components, like accounting, pricing, admission control, and scheduling. The role of the green policies is to implement such strong administrator decisions at the Cloud level. For example, the administrator can establish a power budget per day or per user. The *green policies* module will reject any reservation request that exceeds the power budget limit. This mechanism is similar to a green SLA between the user and the provider.

Green accounting will give to "green" users (the users that accept to delay their reservation in order to aggregate it with others to save energy) credits which are used to give more priority to the requests of these users when a burst of reservation requests arrives. A business model can also lean on this accounting to encourage users to be energy aware.

5 Scenario and Experimental Results

5.1 Experimental Scenario

This section describes the first results obtained with a prototype of GOC. Our experimental platform consists of HP Proliant 85 G2 Servers (2.2 GHz, 2 dual core CPUs per node) with XenServer 5.0[2] [3, 1] on each node.

The following experiments aim to illustrate the working of GOC infrastructure in order to highlight and to compare the energy consumption induced by a Cloud infrastructure with different management schemes. Our experimental platform con-

[2] XenServer is a cloud-proven virtualisation platform that delivers the critical features of live migration and centralised multi-server management.

sists of two identical Cloud nodes, one resource manager that is also the scheduler, and one energy data collector. All these machines are connected to the same Ethernet router. In the following we will call 'job' a reservation made by a user to have the resources at the earliest possible time. When a user submits a reservation, she specifies the length in time and the number of resources required. Although this reservation mechanism may look unusual, it is likely to be used by next generation Clouds where frameworks would support these features to help cloud providers avoid overprovisioning resources. Having scheduling reservations with limits, such as a time duration, will help Cloud managers to manage their resources in a more energy-efficient way. This is, for instance, the case of a user with budget constraints and whose reservations will have a defined time-frame that reflects how much the user is willing to pay for using the resources. In addition, this reflects a scenario where service clouds with long-live applications have their resource allotments adapted according to changes in cost conditions (*e.g.* cost changes in electricity) and scheduled maintenances.

Our job arrival scenario is as follows:

- $t = 10$: 3 jobs of length equals to 120 seconds each and 3 jobs of length 20 seconds each;
- $t = 130$: 1 job of length 180 seconds;
- $t = 310$: 8 jobs of length 60 seconds each;
- $t = 370$: 5 jobs of length 120 seconds each, 3 jobs of length 20 seconds each and 1 job of length 120 seconds, in that order.

The reservations' length is short in order to keep the experiment and the graph representation readable and understandable. Each reservation is a computing job with a *cpuburn* running on the VM. We have seen that the boot and the shutdown of a VM are less consuming than a *cpuburn* and that an idle VM does not consume any noticeable amount of energy (see Figure 2). Hence, we will just include the *cpuburn* burn phase in the graphs in order to reduce their length and improve their readability (VMs are booted before the experiment start and halted after the end of the experiment).

These experiments, although in a small scale for clarity sake, represent the most unfavourable case in terms of energy consumption as *cpuburn* fully uses the CPU which is the most energy consuming component of physical nodes. Moreover, as CPU is a great energy consuming component, *cpuburn* jobs are accurately visible on the power consumption curves: clear steps are noticeable for each added VM (as previously noticed in Figure 2).

Each node can host up to seven VMs. All the hosted VMs are identical in terms of memory and CPU configuration and they all host a *debian etch* distribution. As our infrastructure does not depend on any particular resource manager, we do not change the scheduling of the reservations and the assignment of the virtual machines to physical machines. The only exception is when a physical node is off, where we then attribute its jobs to the awoken nodes if they can afford it; otherwise we switch it on.

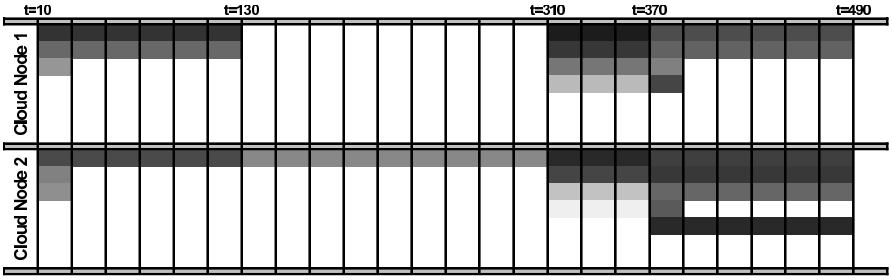


Fig. 9: Gantt chart for the round-robin scheduling.

In order to validate our framework and to prove that it achieves energy saving regardless of the underlying scheduler, we have studied two different schedulings:

- *round-robin*: first job is assigned to the first Cloud node, the second job to the second node, and so on. When all the nodes are idle, the scheduler changes their order (we do not always attribute the first job in the queue to the first node). The behaviour of this scheduling mechanism with the previously defined job arrival scenario is shown in Figure 9. This is a typical distributed-system scheduling algorithm.
- *unbalanced*: the scheduler puts as many jobs as possible on the first Cloud node and, if there are still jobs left in the queue, it uses the second node, and so on (as before, when all the nodes are idle, we change the order to balance the roles). We can see this scheduling with the previously defined job arrival scenario in Figure 10. This scheduling is broadly used for load consolidation.

Fig. 10: Gantt chart for the unbalanced scheduling.

These two scheduling algorithms are well-known and widely used in large-scale distributed system schedulers. For each of these algorithms, we use four scenarios to see the difference between our VM management scheme and a basic one. The four scenarios are as follows:

- *basic*: no changes are made, this scenario represents the Clouds with no power management.
- *balancing*: migration is used to balance the load between the Cloud nodes. This scenario presents the case of a typical load balancer which aims to limit node failures and heat production.
- *on/off*: unused nodes are switched off. This is the scenario with a basic power management;
- *green*: we switch off the unused nodes and we use migration to unbalance the load between Cloud nodes. This allow us to aggregate the load on some nodes and switch off the other ones. This is the scenario that corresponds to GOC.

5.2 Results

For each scheduling, we have run the four scenarios, one at a time, on our experimental platform and we have logged the energy consumption (one measure per node and per second at the precision of 0.125 Watts). A more extensive discussion on the results is available in previous work [21].

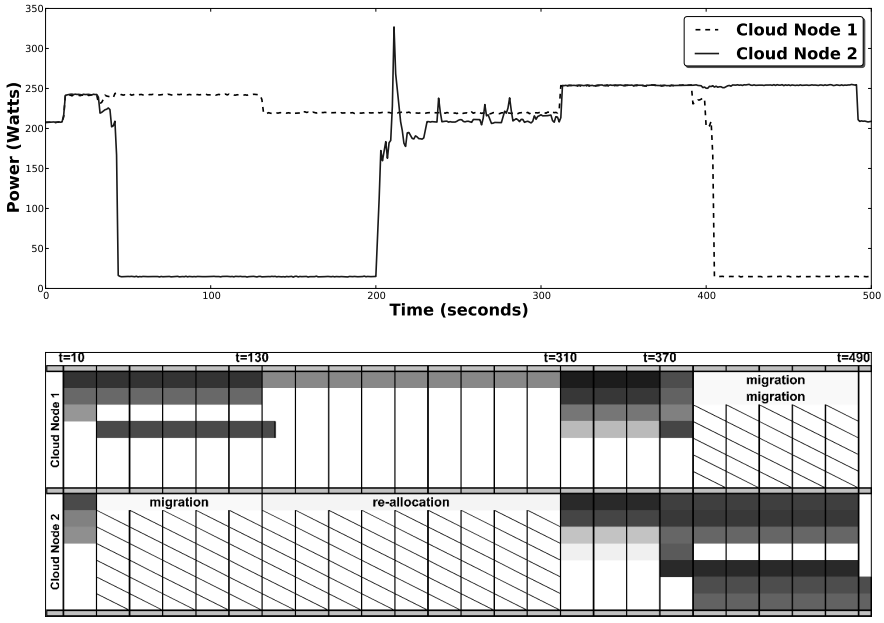


Fig. 11: Green scenario with round-robin scheduling.

Figure 11 shows the course of experiment for the two nodes with the round-robin scheduling applied to the green scenario. The upper part of the figure shows the energy consumption of the two nodes during the experiment while the lower part presents the Gantt chart (time is in seconds).

At time $t = 30$, the second job (first VM on Cloud node 2) is migrated to free Cloud node 1 and then to switch it off. This migration does not occur before $T_a = 20$ seconds. Some small power peaks are noticeable during the migration. This confirms the results of Section 3.3 stating that migration is not really costly if it is not too long.

This migration leads to the re-allocation of the job starting at $t = 130$ on Cloud node 1 since Cloud node 2 has been switched off and Cloud node 1 is available. At $t = 200$, Cloud node 2 is booted to be ready to receive the jobs starting at $t = 310$. During the boot, an impressive consumption peak occurs. It corresponds to the physical start of all the fans and the initialisation of all the node components. Yet, this peak is more than compensated by the halting of the node ($P_{OFF} = 20$ Watts) during the time period just before the boot since the node inactivity time was greater than T_s (defined in Section 4.3).

During the seventh job, one can notice that a running VM with a *cpuburn* inside consumes about 10 Watts which represents about 5% of the idle consumption of the Cloud node. At time $t = 390$, two new VM migrations are performed, and they draw small peaks on the power consumption curves. From that time, Cloud node 1 is switched off and Cloud node 2 has 6 running VMs. We observe that the fifth and the sixth VMs cost nothing (no additional energy consumption compared to the period with only four VMs) as explained in Section 3.2 as this Cloud node has four cores. This experiment shows that GOC greatly takes advantage of the idle periods by using its green enforcement solutions: VM migration and shut down. Job re-allocation also allows to avoid on/off cycles. Significant amounts of energy are saved.

The green scenario with unbalanced scheduling is presented in Figure 12. For the green scenario, the unbalanced scheduling is more in favour of energy savings. Indeed, two migrations less are needed than with the round-robin scheduling, and all the first burst jobs are allocated to Cloud node 1 allowing Cloud node 2 to stay off. In fact, this is the general case for all the scenarios (Figure 13): the unbalanced scheduling is more energy-efficient since it naturally achieves a better consolidation on fewer nodes.

Summarised results of all the experiments are shown in Figure 13. As expected, the green scenario which illustrates GOC behaviour is the less energy consuming for the both scheduling scenarios. The balancing scenario is more energy consuming than the basic scheme for unbalanced scheduling. This phenomenon is due to numerous migrations for the balancing scenario that cancel out the benefits of these migrations.

The noticeable figure is that the green scenario with unbalanced scheduling consumes 25% less electricity than the basic scenario which shows the current Clouds administration. This figure obviously depends on the Cloud usage.

However, it shows that great energy savings are achievable with minor impacts on the utilisation: small delays occur due to migrations (less than 5 seconds for

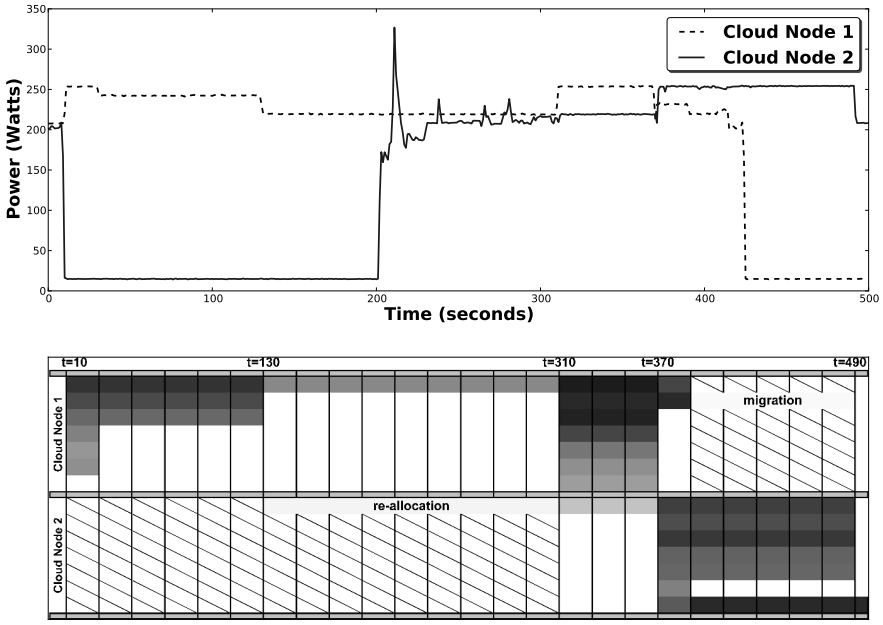


Fig. 12: Green scenario with unbalanced scheduling.

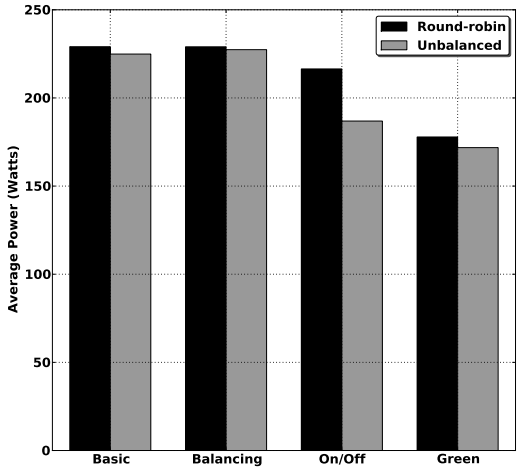


Fig. 13: Comparison results.

these experiments) and to physical node boots (2 minutes for our experimental platform nodes). These delays can be improved by using better VM live-migration techniques [13], and by using faster booting techniques (like suspend to disk and suspend to RAM techniques).

6 Conclusion

As Clouds are more and more broadly used, the Cloud computing ecosystem becomes a key challenge with strong repercussions. It is therefore urgent to analyse and to encompass all the stakeholders related to the Cloud's energy usage in order to design adapted framework solutions. It is also essential to increase the energy-awareness of users in order to reduce their CO_2 footprint induced by their utilisations of Cloud infrastructures. This matter lead us to measure and to study the electrical cost of basic operations concerning the VM management in Clouds, such as the boot, the halt, the inactivity and the launching of a sample cpuburn application. These observations show that VMs do not consume energy when they are idle and that booting and halting VMs produce small power peaks, but are not really costly in terms of time and energy.

As Clouds are on the way to becoming an essential element of future Internet, new technologies have emerged for increasing their capacity to provide on-demand XaaS (everything as a service) resources in a pas-as-you-use fashion. Among these new technologies, live migration seems to be a really promising approach allowing on-demand resource provisioning and consolidation. We have studied the contribution that the use of this technique could constitute in terms of energy efficiency. VM live migration, although its performance can be improved, is reliable, fast, and requires little energy compared to the amount it can save.

These first measurements have allowed us to propose an original software framework: the Green Open Cloud (GOC) which aims at controlling and optimising the energy consumed by the overall Cloud infrastructure. This energy management is based on different techniques:

- energy monitoring of the Cloud resources with energy sensor for taking efficient management decisions;
- displaying the energy logs on the Cloud portal to increase the energy-awareness;
- switching off unused resources to save energy;
- utilisation of a proxy to carry out network presence of switched-off resources and thus provide inter-operability with any kind of RMS;
- use of VM migration to consolidate the load in fewer resources;
- prediction of the next resource usage to ensure responsiveness;
- giving green advice to users in order to aggregate resources' reservations.

The GOC framework embeds features that, in our opinion, will be part of the next-generation Clouds, such as advance reservations, which enable a more flexible

and planned resource management; and VM live-migration. which allows for great workload consolidation.

Further, the GOC framework was the subject of an experimental validation. The validation is made with two different scheduling algorithms to prove GOC's adaptivity to any kind of Resource Management System (RMS). Four scenarios are studied to compare the GOC behaviour with basic Cloud's RMS workings. Energy measurements on these scenarios show that GOC can save up to 25% of the energy consumed by a basic Cloud resource management. These energy savings are also reflected in the operational costs of the Cloud infrastructures.

This promising work shows that important energy savings and thus CO_2 footprint reductions are achievable in future Clouds at the cost of minor performance degradations. There is still room for improvement to increase the user's energy-awareness which is the main leverage to trigger a real involvement from the Cloud providers and designers in order to reduce the electricity demand of Clouds and contribute to a more sustainable ICT industry.

References

1. Citrix xenserver. URL `http://citrix.com/English/ps2/products/product.asp?contentID=683148`
2. Make it green - cloud computing and its contribution to climate change. Greenpeace international (2010)
3. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: 19th ACM Symposium on Operating Systems Principles (SOSP '03), pp. 164–177. ACM Press, New York, USA (2003). DOI http://doi.acm.org/10.1145/945445.945462
4. Box, G.E.P., Jenkins, G.M., Reinsel, G.C.: Time Series Analysis: Forecasting and Control, 3rd edn. Prentice-Hall International, Inc. (1994)
5. Chase, J.S., Anderson, D.C., Thakar, P.N., Vahdat, A.M., Doyle, R.P.: Managing energy and server resources in hosting centers. In: 18th ACM Symposium on Operating Systems Principles (SOSP '01), pp. 103–116. ACM Press, Banff, Alberta, Canada (2001)
6. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, pp. 273–286. Berkeley, CA, USA (2005)
7. Da-Costa, G., Gelas, J.P., Georgiou, Y., Lefèvre, L., Orgerie, A.C., Pierson, J.M., Richard, O., Sharma, K.: The green-net framework: Energy efficiency in large scale distributed systems. In: HPPAC 2009: High Performance Power Aware Computing Workshop in conjunction with IPDPS 2009. Roma, Italy (2009)
8. Doyle, R.P., Chase, J.S., Asad, O.M., Jin, W., Vahdat, A.M.: Model-based resource provisioning in a Web service utility. In: 4th Conference on USENIX Symposium on Internet Technologies and Systems (USITS'03), pp. 5–5. USENIX Association, Berkeley, CA, USA (2003)
9. Fan, X., Weber, W.D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. In: ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture, pp. 13–23. New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1250662.1250665
10. Fontán, J., Vázquez, T., Gonzalez, L., Montero, R.S., Llorente, I.M.: OpenNEbula: The open source virtual machine manager for cluster computing. In: Open Source Grid and Cluster Software Conference – Book of Abstracts. San Francisco, USA (2008)

11. Hamm, S.: With Sun, IBM Aims for Cloud Computing Heights (26 March 2009). URL http://www.businessweek.com/magazine/content/09_14/b4125034196164.htm?chan=magazine+channel_news
12. Harizopoulos, S., Shah, M.A., Meza, J., Ranganathan, P.: Energy efficiency: The new holy grail of data management systems research. In: Fourth Biennial Conference on Innovative Data Systems Research (CIDR) (2009). URL http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_112.pdf
13. Hirofuchi, T., Nakada, H., Ogawa, H., Itoh, S., Sekiguchi, S.: A live storage migration mechanism over wan and its performance evaluation. In: VTDC '09: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing, pp. 67–74. ACM, New York, NY, USA (2009). DOI http://doi.acm.org/10.1145/1555336.1555348
14. Hotta, Y., Sato, M., Kimura, H., Matsuoka, S., Boku, T., Takahashi, D.: Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster. IPDPS 2006 (2006). DOI 10.1109/IPDPS.2006.1639597
15. Iosup, A., Dumitrescu, C., Epema, D., Li, H., Wolters, L.: How are real grids used? the analysis of four grid traces and its implications. In: 7th IEEE/ACM International Conference on Grid Computing (2006)
16. Jung, G., Joshi, K.R., Hiltunen, M.A., Schlichting, R.D., Pu, C.: A cost-sensitive adaptation engine for server consolidation of multitier applications. In: 10th ACM/IFIP/USENIX International Conference on Middleware (Middleware 2009), pp. 1–20. Springer-Verlag New York, Inc., New York, NY, USA (2009)
17. Kalman, R.E.: A new approach to linear filtering and prediction problems. Transactions of the ASME – Journal of Basic Engineering **82**(Series D), 35–45 (1960)
18. Kalyvianaki, E., Charalambous, T., Hand, S.: Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters. In: 6th International Conference on Autonomic Computing (ICAC 2009), pp. 117–126. ACM, New York, NY, USA (2009). DOI http://doi.acm.org/10.1145/1555228.1555261
19. Kim, M., Noble, B.: Mobile network estimation. In: 7th Annual International Conference on Mobile Computing and Networking (MobiCom 2001), pp. 298–309. ACM, New York, NY, USA (2001). DOI http://doi.acm.org/10.1145/381677.381705
20. Kusic, D., Kephart, J.O., Hanson, J.E., Kandasamy, N., Jiang, G.: Power and performance management of virtualized computing environments via lookahead control. In: 5th International Conference on Autonomic Computing (ICAC 2008), pp. 3–12. IEEE Computer Society, Washington, DC, USA (2008). DOI http://dx.doi.org/10.1109/ICAC.2008.31
21. Lefèvre, L., Orgerie, A.C.: Designing and evaluating an energy efficient cloud. The Journal of SuperComputing **51**(3), 352–373 (2010)
22. Liu, J., Zhao, F., Liu, X., He, W.: Challenges towards elastic power management in Internet data centers. In: 29th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW 2009), pp. 65–72. IEEE Computer Society, Washington, DC, USA (2009). DOI http://dx.doi.org/10.1109/ICDCSW.2009.44
23. Miyoshi, A., Lefurgy, C., Van Hensbergen, E., Rajamony, R., Rajkumar, R.: Critical power slope: understanding the runtime effects of frequency scaling. In: ICS '02: Proceedings of the 16th international conference on Supercomputing, pp. 35–44. ACM, New York, NY, USA (2002). DOI http://doi.acm.org/10.1145/514191.514200
24. Moore, J., Chase, J., Ranganathan, P., Sharma, R.: Making scheduling "cool": Temperature-aware workload placement in data centers. In: USENIX Annual Technical Conference (ATEC 2005), pp. 5–5. USENIX Association, Berkeley, CA, USA (2005)
25. Nathuji, R., Schwan, K.: VirtualPower: Coordinated power management in virtualized enterprise systems. In: 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP 2007), pp. 265–278. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1294261.1294287
26. Nurmi, D., Wolski, R., Crzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: Eucalyptus: A technical report on an elastic utility computing architecture linking your programs to useful systems. Technical report 2008-10, Department of Computer Science, University of California, Santa Barbara, California, USA (2008)

27. Orgerie, A.C., Lefèvre, L., Gelas, J.P.: Chasing gaps between bursts: Towards energy efficient large scale experimental grids. In: PDCAT 2008: The Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 381–389. Dunedin, New Zealand (2008)
28. Orgerie, A.C., Lefèvre, L., Gelas, J.P.: Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems. In: 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS), pp. 171–178. Melbourne, Australia (2008)
29. Patterson, M., Costello, D., Grimm, P., Loeffler, M.: Data center TCO: a comparison of high-density and low-density spaces. In: Thermal Challenges in Next Generation Electronic Systems (THERMES 2007) (2007). URL `http://isdlibrary.intel-dispatch.com/isd/114/datacenterTCO_WP.pdf`
30. Sharma, R., Bash, C., Patel, C., Friedrich, R., Chase, J.: Balance of power: Dynamic thermal management for internet data centers. IEEE Internet Computing **9**(1), 42–49 (2005). DOI http://doi.ieeecomputersociety.org/10.1109/MIC.2005.10
31. Silicon Valley Leadership Group: Data Center Energy Forecast. White Paper (2008). URL `svlg.org/campaigns/datacenter/docs/DCEFR_report.pdf`
32. Singh, T., Vara, P.K.: Smart metering the clouds. IEEE International Workshops on Enabling Technologies **0**, 66–71 (2009). DOI http://doi.ieeecomputersociety.org/10.1109/WETICE.2009.49
33. Snowdon, D.C., Ruocco, S., , Heiser, G.: Power Management and Dynamic Voltage Scaling: Myths and Facts. In: Proceedings of the 2005 Workshop on Power Aware Real-time Computing (2005)
34. Srikantaiah, S., Kansal, A., Zhao, F.: Energy aware consolidation for cloud computing. In: Proceedings of HotPower '08 Workshop on Power Aware Computing and Systems (2008). URL `http://www.usenix.org/events/hotpower08/tech/full_papers/srikantaiah/srikantaiah/_html/`
35. Subramanyam, S., Smith, R., van den Bogaard, P., Zhang, A.: Deploying Web 2.0 applications on Sun servers and the opensolaris operating system. Sun BluePrints 820-7729-10, Sun Microsystems (2009)
36. Talaber, R., Brey, T., Lamers, L.: Using Virtualization to Improve Data Center Efficiency. Tech. rep., The Green Grid (2009)
37. Tatezono, M., Maruyama, N., Matsuoka, S.: Making wide-area, multi-site MPI feasible using Xen VM. In: Workshop on Frontiers of High Performance Computing and Networking (held with ISPA 2006), *LNCS*, vol. 4331, pp. 387–396. Springer, Berlin/Heidelberg (2006)
38. Travostino, F., Daspit, P., Gommans, L., Jog, C., de Laat, C., Mambretti, J., Monga, I., van Oudenaarde, B., Raghunath, S., Wang, P.Y.: Seamless live migration of virtual machines over the man/wan. Future Gener. Comput. Syst. **22**(8), 901–907 (2006). DOI http://dx.doi.org/10.1016/j.future.2006.03.007
39. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., Wood, T.: Agile dynamic provisioning of multi-tier internet applications. ACM Transactions on Autonomous and Adaptive Systems **3**(1), 1–39 (2008). DOI http://doi.acm.org/10.1145/1342171.1342172
40. Verma, A., Ahuja, P., Neogi, A.: pMapper: Power and migration cost aware application placement in virtualized systems. In: ACM/IFIP/USENIX 9th International Middleware Conference (Middleware 2008), pp. 243–264. Springer-Verlag, Berlin, Heidelberg (2008). DOI http://dx.doi.org/10.1007/978-3-540-89856-6_13

Glossary

Cloud Computing a business model that provides utility Computing services and/or SaaS services.

Grid Computing distributed computing that enables IT scalability and flexibility, mainly focusing on large-scale problems.

Platform as a service Pay-per-Use for network based delivery of a computing platform and a related solution stack as a service.

Service Orientation a design paradigm that specifies the creation of automation logic in the form of services. It is applied as a strategic goal in developing a service-oriented architecture (SOA).

Software as a Service Pay-per-Use for network based software applications services.

Utility Computing Pay-per-Use for network based Compute and Storage services.

Virtualization a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources.

Index