Metadata Management in Multi-Grids and Multi-Clouds

Daniel Espling[*]

LICENTIATE THESIS, SEPTEMBER 2011
DEPARTMENT OF COMPUTING SCIENCE
UMEÅ UNIVERSITY
SWEDEN

[*] Previously Henriksson.

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

espling@cs.umu.se

ISBN 978-91-7459-281-8
ISSN 0348-0542
UMINF 11.08

Printed by Print & Media, Umeå University, 2011

Abstract

Grid computing and *cloud computing* are two related paradigms used to access and use vast amounts of computational resources. The resources are often owned and managed by a third party, relieving the users from the costs and burdens of acquiring and managing a considerably large infrastructure themselves. Commonly, the resources are either contributed by different stakeholders participating in shared projects (grids), or owned and managed by a single entity and made available to its users with charging based on actual resource consumption (clouds). Individual grid or cloud sites can form collaborations with other sites, giving each site access to more resources that can be used to execute *tasks* submitted by users. There are several different models of collaborations between sites, each suitable for different scenarios and each posing additional requirements on the underlying technologies.

Metadata concerning the status and resource consumption of tasks are created during the execution of the task on the infrastructure. This metadata is used as the primary input in many core management processes, e.g., as a base for accounting and billing, as input when prioritizing and placing incoming task, and as a base for managing the amount of resources allocated to different tasks.

Focusing on management and utilization of metadata, this thesis contributes to a better understanding of the requirements and challenges imposed by different collaboration models in both grids and clouds. The underlying design criteria and resulting architectures of several software systems are presented in detail. Each system addresses different challenges imposed by cross-site grid and cloud architectures:

- The LUTSfed approach provides a lean and optional mechanism for filtering and management of usage data between grid or cloud sites.
- An accounting and billing system natively designed to support cross-site clouds demonstrates usage data management despite unknown placement and dynamic task resource allocation.
- The FSGrid system enables fairshare job prioritization across different grid sites, mitigating the problems of heterogeneous scheduling software and local management policies.

The results and experiences from these systems are both theoretical and practical, as full scale implementations of each system has been developed and analyzed as a part of this work. Early theoretical work on structure-based service management forms a foundation for future work on structured-aware service placement in cross-site clouds.

Populärvetenskaplig Sammanfattning

Grid computing och *cloud computing* är två besläktade metodiker för att komma åt och nyttja stora mängder datorresurser, exempelvis för att göra omfattande beräkningar och simuleringar eller till lagring av väldigt stora mängder data. Datorresurserna ägs och underhålls ofta av en tredje part, vilket besparar användarna kostnaderna och mödan att införskaffa och underhålla den stora infrastrukturen själva, speciellt som den stora mängden datorkraft oftast bara behövs under kortare perioder. Vanligtvis är resurserna antingen ägda av flera oberoende parter som deltar i gemensamma projekt (grid), eller ägda av en enda organisation och görs tillgängliga för allmänheten (eller en begränsad mängd användare) för att sedan debitera användare för datorkraften de faktiskt använder (clouds). Enskilda grids eller clouds kan samarbeta med andra aktörer för att få tillgång till än större mängder resurser som kan användas till att köra jobb åt användarna. Det finns flera olika samarbetsmodeller mellan aktörer som lämpar sig för olika tillfällen, och varje modell medför ytterligare krav på den underliggande tekniken.

När jobb körs på infrastrukturen skapas metadata, information om statusen hos jobbet och mängden resurser som förbrukas när jobbet körs. Dessa metadata är det huvudsakliga underlaget för flera interna processer i infrastrukturen. Exempelvis används det som bas för fakturering, som beslutsunderlag för att välja vilken ordning man ska prioritera jobb och som en indikation för när mängden resurser som tilldelats ett jobb behöver ökas eller minskas.

Med fokus på hanteringen och nyttandet av jobbmetadata bidrar denna avhandling till en djupare förståelse för de problem och krav som uppkommer i grids eller clouds som använder datorresurser från flera olika aktörer. Underliggande designkriterier och de resulterande arkitekturerna för flera mjukvarusystem presenteras i detalj. Varje system fokuserar på olika delar av de utmaningar som sammarbetsmodeller för grids och clouds medför:

- LUTSfed bidrar med filtrering och hantering av metadata mellan flera grids och clouds på ett minimalistisk och smidigt sätt.
- Ett system för bokföring och fakturering från grunden designat för att stödja flera clouds demonstrerar hur användningsdata kan hanteras utan kännedom om var jobben körs eller vetskap om hur mycket resurser jobbet kräver.

- FSGrid möjligör prioritering baserad på tidigare förbrukningsdata på ett enhetligt sätt över flera grids, oavsett skillnader i underliggande mjukvaror eller lokala policies.

Resultaten och erfarenheterna från dessa system är inte enbart teoretiska, eftersom fullskaliga implementationer av samtliga system har utvecklats och analyserats som en del av det här arbetet. Tidiga teoretiska resultat med fokus på placering av jobb i clouds där den interna strukturen hos jobbet tas i beaktning skapar en grund för vidare arbete inom ämnet.

Preface

This thesis contains an introduction to grid and cloud computing, with focus on metadata management, and the below listed papers. The author changed surname from Henriksson to Espling just prior to printing this thesis, which is why the articles included in this thesis are printed under a different name than the thesis itself.

Paper I E. Elmroth and D. Henriksson. Distributed Usage Logging for Federated Grids. *Future Generations Computer Systems*, 26(8):1215–1225, 2010.

Paper II E. Elmroth, F. Galán, D. Henriksson, and D. Perales. Accounting and Billing for Federated Cloud Infrastructures. In *GCC '09: Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing*, pages 268–275, Washington, DC, USA, 2009. IEEE Computer Society.

Paper III L. Larsson, D. Henriksson, and E. Elmroth. Scheduling and Monitoring of Internally Structured Services in Cloud Federations. In *Proceedings of IEEE ISCC 2011*, pages 173–178, 2011.

Paper IV P-O. Östberg, D. Henriksson, and E. Elmroth. Decentralized, scalable, Grid Fairshare Scheduling (FSGrid). 2011. Submitted.

This research was conducted using the resources of the High Performance Computing Center North (HPC2N) and the UMIT research lab. Financial support has been provided by The Swedish Research Council (VR) under contract 621-2005-3667, by the European Community's Seventh Framework Programme ([FP7/2001-2013]) under grant agreement no. 215605 (RESERVOIR) and no. 257115 (OPTIMIS).

In addition to the publications included in the thesis, the following papers on related subjects has also been produced in the context of this work:

- M. Lindner, F. Galán, C. Chapman, S. Clayman, D. Henriksson, and E. Elmroth. The Cloud Supply Chain: A Framework for Information, Monitoring, Accounting and Billing. In *2nd International ICST Conference on Cloud Computing (CloudComp 2010)*.

- M. B. Yehuda, O. Biran, D. Breitgand, K. Meth, B. Rochwerger, E. Salant, E. Silvera, S. Tal, Y. Wolfsthal, J. Cáceres, J. Hierro, W. Emmerich, A. Galis, L. Edblom, E. Elmroth, D. Henriksson, F. Hernández, J. Tordsson, A. Hohl, E. Levy, A. Sampaio, B. Scheuermann, M. Wusthoff, J. Latanicki, G. Lopez,

J. Marin-Frisonroche, A. Dörr, F. Ferstl, S. Beco, F. Pacini, I. Llorente, R. Montero, E. Huedo, P. Massonet, S. Naqvi, G. Dallons, M. Pezzé, A. Puliato, C. Ragusa, M. Scarpa, and S. Muscella. RESERVOIR - an ICT infrastructure for reliable and effective delivery of services as utilities. Technical report, IBM Haifa Research Laboratory, 2008.

- G. Katsaros, G. Gallizo, R. Kübert, T. Wang, J. O. Fito, and D. Henriksson. A Multi-level Architecture for Collecting and Managing Monitoring Information in Cloud Environments. In *CLOSER 2011 : International Conference on Cloud Computing and Services Science (CLOSER)*, 2011. Accepted for publication.

Acknowledgments

First and foremost, I would like to thank my supervisor Erik Elmroth for creating (and maintaining) a pleasant, supportive, and inspiring research environment, and for always finding the time despite being a resource constantly subject to overbooking. I am also very grateful for the help and feedback given by my co-supervisor, Johan Tordsson, who took the time to give feedback on this thesis in mid July despite being on vacation and despite Tour de France running on TV.

A big thank you to all collaborators, colleagues in and outside our group, and coauthors of papers both within and outside the bounds of this thesis. You are too numerous to be mentioned by name, but interacting with the lot of you and sharing your views of things to solve shared problems is what makes this job interesting. We are also blessed with a very competent, kind, and understanding administrational staff, both for technical and non-technical tasks. Thank you for making our everyday working lives easier and for never backing down from challenges such as installing software we produce, or sorting my post-laundry traveling receipts.

A special thanks to Lars Larsson, my constant 2*vX* ally. Not only for daily company, support, and interesting discussions, but also for teaching me to leverage obscure tools and features, and for explaining countless times why things like *gqap* are perfectly sane commands to learn by heart.

Last but definitely not least I would like to thank my closest family and my friends for providing an outstanding environment to grow up, live, and hopefully grow old in. To my recently wedded wife Maria Espling, with whom I share everything (including the hassle of changing name halfway through a PhD): *Du är mitt guld också.*

Umeå, September 2011
Daniel Espling

Contents

Chapter 1

Introduction

Computing capacity available as a utility similar to water or electricity has been a vision for a very long time, with the predictions of John McCarty dating from the early sixties often seen as the starting point [62, 64]. Fifty years later there have been several incarnations of this paradigm, with the same underlying goal of computing capacity as a utility. Most often, the new paradigm does not entirely overlap with the previous paradigms in scope, leaving niches for several generations of paradigms to coexist.

Two of the most recent paradigms for computing as a utility are *grid computing* and *cloud computing*. We refer to the paradigms at large simply as *grids* and *clouds*, and use the terms *site* or *provider* to emphasize a single supplier in either paradigm. Work units sent to a grid are usually denoted *jobs* while those sent to a cloud are called *services*[1]. As cloud computing is a quite wide term (see Chapter 3), a cloud *service* can denote several different things. As most of this thesis focus on infrastructure management, we use service to denote self-contained work units supplied to infrastructure providers for execution. We also use the term *task*to denote both grid jobs and cloud services, and each term separately when referring only to either.

Grids and clouds are both fundamentally ways to group existing (heterogeneous) computer resources into an abstract pool of resources, and making those resources available to users as a virtual coherent infrastructure. Starting out with similar objectives, grids have evolved into reliable, high performing platforms mostly used for large-scale scientific computing while clouds has emerged as a remote hosting and execution option for many different kinds of software. Chapter 2 and Chapter 3 describe these paradigms in more detail.

Other relevant paradigms are, e.g., High Performance Computing (HPC) [44] and High Throughput Computing (HTC) [111]. HPC systems focus on running parallel jobs on centralized, dedicated hardware with very high performance in terms of, e.g., computational speed and network latency. HTC on the other hand focuses on maximizing the use of distributed, widely heterogeneous, and

[1]Not to be confused with Web Services [30] as a technology.

unreliable resources not for the sake of a single job but for the general system as a whole. Even though, from a management perspective, HPC and HTC avoids many of the challenges of grids and clouds covered by this thesis, concepts such as those in Paper I (accounting data management) and Paper IV (decentralized fairshare scheduling) can be applied to HPC and HTC environments as well.

Individual grids and clouds can be joined into even bigger pools of resources through collaborations. These multi-grid and multi-cloud environments pose additional challenges for the management of submitted tasks, and several different collaboration models with unique challenges exists [55, 57]. One such collaboration model is *federations* of grids or clouds, where a single grid or cloud may utilize resources from other sites, commonly as part of bilateral resource exchange agreements. For grids, large projects such as the Large Hadron Collider (LHC) [108] has outgrown the capacity of any single grid and require cross-grid solutions to cope with the high resource demand. Similarly, clouds form collaborations to cope with surges in demand when local resources are not sufficient, giving the impression of clouds as endless pools of resources. In some cases, the collaborating cloud may in turn outsource the execution to a third cloud site, creating a chain of delegation from the originating site to the site where the task is finally executed. Clients for grids and clouds should be kept unaware and unconcerned about whether the infrastructure is part of a collaboration or not, and will normally not be aware of on which collaborating site a submitted task is finally executed (as long as the job does not have explicit restrictions on placement). Therefore, the underlying infrastructure itself must deal with any heterogeneity or additional complexity imposed by the collaborative environment, for example the task metadata management.

Metadata concerning, e.g., the resource consumption or duration of a task are collected during (or after) the execution of a task. This metadata has to be collected and managed equally regardless of if the task executes locally or at a collaborating site, as the data is commonly used as basis for many internal processes in both grids and clouds. The process of collecting, sharing, and managing run time information about a task is called *monitoring*. Grids normally only use monitoring information regarding the state of physical resources, and utilize job metadata generated upon job completion for tasks such as accounting, billing, and job scheduling. Clouds typically rely solely on run time monitoring data for internal management processes, as cloud services does not have a fixed execution time.

The focus of thesis is how to collect, manage, and utilize task metadata in different collaboration models of grids and clouds. The thesis investigates how these fundamental tasks are affected by the barriers imposed by collaborations such as federations, e.g., technical heterogeneity, distributed and (site-wise) self-centric decision making, and incomprehensive information on the state and availability of remote resources. Papers I and II focus primarily on the collection and management of task metadata, while papers III and IV focus on how to utilize the task metadata for resource allocations in clouds and grids.

The following summarizing chapters presented prior to the papers provides

a general introduction and context to topics relevant to the presented papers: Chapter 2 presents a basic overview of grid computing.Chapter 3 describes cloud computing, including a detailed explanation of the infrastructure management of clouds and several different collaboration scenarios. Chapter 4 presents an overview of task metadata management in both grids and clouds. Papers are summarized in Chapter 5 and potential directions for future work are outlined in Chapter 6 before the bibliography finalizes the summarizing chapters.

Chapter 2

Grid Computing

The foundation of an open networking structure that would later emerge into the Internet was laid by the National Science Foundation (NSF) [123] back in 1986, when the NFSNET backbone was built to connect five supercomputers in the U.S. [62, 107]. Twenty five years later the Internet has evolved into a general utility used by more than two billion people [119]. Meanwhile, grid computing [62] has emerged as a technology and paradigm focusing on the original intent of the Internet – interconnecting resources to form supercomputers.

The analogy between the Internet and grid computing runs deep. The Internet started out as several isolated networks (for example CSNET [35] and ARPANET [2]) only available to specific research communities [107]. Since then, it has evolved into a ubiquitous, unified, and commonly available communications utility. Grid computing stems from the vision of offering computer resources as easily and transparently as electricity using the power grid (and hence the name), while in reality the concept of *The Grid* is still at the stage of early Internet; existing grids are isolated networks targeting specific communities, primarily used for large-scale research projects.

Grid computing as a concept has grown vast enough to encompass many different tools for many different tasks, becoming a group of related technologies rather than a single unified utility. This, and the fact that there is no absolute definition distinguishing grids from other distributed environments, leads to some confusion on what should be considered a grid. Among many definitions [21, 36, 155], the most commonly used definition by Foster [60] comes in the form of a three point checklist, defining grids as systems that: "coordinates resources that are not subject to centralized control ...", "using standard, open, general-purpose protocols and interfaces ...", "to deliver nontrivial qualities of service".

Foster's definition is widely accepted but not standardized, and there are major grid efforts (such as the LHC Computing Grid (LCG) [97]) that groups resources under centralized control while still being referred to as a grid. The view on grids underlying the work presented in this thesis is very similar to Foster's definition, with emphasis on decentralized control of resources and

autonomy of participating sites.

Since the initial vision of offering general-purpose computational capacity as a utility, grid computing has evolved into a tool mostly used to enable infrastructure for large-scale scientific projects, such as the Large Hadron Collider (LHC) [108], the World-wide Telescope [159], and the Biomedical Informatics Research Network [73]. In many cases, grids are not only means to share raw computational resources but also makes it possible to share data from important scientific instruments. The project-oriented business model, technical problems (often related to software dependencies), and interoperability issues are a some reasons why the use of grid resources are mostly restricted to specific scientific communities [11, 64]. For these communities, however, grids have made it possible to address problems previously out of reach in terms of computational resource requirements or available scientific tools. A comprehensive overview of grid computing and its implications and uses in several fields (bioinformatics, medicine, astronomy, etc.) is given by Foster and Kesselman [62]. Although this book dates from 2004, the conceptual aspects of grids have not changed notably since.

2.1 Grid as an Infrastructure

The overall purpose of grid computing is to interconnect resources which may be owned by different actors in different countries, have different physical characteristics (CPU frequency, CPU architecture, network bandwidth, disk space, etc.) and run different operating systems and software stacks. These resources are consumed by users commonly organized in collaborating scientific communities, Virtual Organizations [63].

A wide variety of grid middlewares including [8, 13, 47, 61, 97, 156, 161, 167] are used as intermediate software layers for job submission and job management in grids. The vast set of different middlewares has created interoperability problems between the middlewares themselves [55], creating an additional niche for software to ease the burden to work with different middlewares [5, 51, 70, 144, 170].

Grid jobs can normally be seen as a self-contained bundle of computational jobs and input data which can be executed independently across different nodes to generate a set of output data. The jobs are batch-oriented and normally no user interaction with the job is required or even possible during execution time, which limits the scope of applications suitable for execution on grids. For non-trivial jobs, however, there is commonly considerable amounts of inter-process communication required during job execution. The Job Submission Description Language (JSDL) [9] is a widely accepted standard for specifying job configuration properties such as hardware requirements, execution deadlines, and sets of input and output file required or generated by the computations.

When running a job on a grid, the first step is to select which of the available resources to execute on. This can either be done manually by the user, or

by the support of a *resource broker* [26, 54, 99]. Once a suitable resource has been selected, the job is submitted for execution to the local scheduler of that resource. Common technologies for local resource scheduling includes Maui [84] and SLURM [181]. In contrast to the local scheduler, the broker does not have full control over the resources and must rely on best-effort scheduling of jobs [146].

The lack of user interactions makes it possible for a grid to schedule (and re-schedule) jobs, as there are normally no strict restrictions on when the job should run. Advance reservations allows users to reserve specific execution times if required, most often at the expense of overall resource utilization due to creation of small unusable gaps prior to the start-time of the reserved jobs [149]. Backfilling techniques [121, 154] are commonly used to increase resource utilization, and may also be used to mitigate the loss of utilization caused by reservations. There are many different strategies to grid job scheduling, some focusing on, e.g, scheduling for the benefit of a single application [18], optimizing the job wait time [77], optimizing the total system throughput [82], avoiding starvation[1], or to offer advance reservations. An early overview and performance comparison of grid scheduling techniques can be found in [79].

Another parameter commonly used in scheduling is *fairness*. The concept, originating from [88], is commonly used in scheduling to take previous consumption and user shares into account, prioritizing jobs for users higher if that user has a lot of unspent shares. There are several approaches to fairshare scheduling in grids, e.g., [38, 43, 45, 49, 94, 96]. The definition of fairness varies between the different approaches, some measuring the total resource utilization, others the number of accepted jobs or the number of missed deadlines per user [129]. All approaches uses some historical utilization data as input in the scheduling process.

A modern batch system scheduler can be configured in many ways to strive towards one or more objectives, normally using weighed combinations of several parameters. The scheduler prioritizes the jobs dynamically and submits jobs for execution on the local resources. After job completion, a *usage record* [112] is generated with metadata concerning the job. This information is subsequently used for internal grid process such as accounting and fairshare scheduling. For more information about metadata management, see Chapter 4.

2.2 Federated Grids

As mentioned in the Internet analogy at the start of the chapter, grids emerged as isolated islands similarly to the early isolated networks now made a part of the unified Internet. The initial vision of grid was a wide spanning resource network functioning as a utility, and there are several efforts to create *federations* of grids [20, 65, 106, 132], where grids unifies (parts of) their resources for

[1] Starvation occurs when some jobs are constantly neglected in favor or other jobs, starving them of resources.

common use while still retaining full control over the local infrastructure. For example, the Swedish and Norwegian national grids [125, 150] are two of the actors contributing resources to the Nordic Data Grid Facility (NDGF) [124] consortium. Even though the resources are acquired, owned, and managed by each national grid, a subset of the jobs executed on these resources are run on the behalf of NDGF. In a federation of grids, each site must remain a fully functional autonomous grid in itself, unlike regular computational resources constituting a normal grid which may rely on common grid functionality in order to function. Therefore, federated grids require fully decentralized, but interoperable, solutions in particular for scheduling and metadata management.

The motivations behind federations of grids are not only technical, but often economical or political to consolidate resources and promote collaborations. For instance, EGEE (originally Enabling Grids for E-science in Europe) project [105] is a series of projects initiated by the European Union to create a wide spanning computational grid infrastructure based mainly on the gLite [104] middleware. European Grid Initiative (EGI) [98] is a substantial European initiative to further unify national grids across Europe, largely continuing on the EGEE effort but with a significant focus on seamless interoperability and integration of several different underlying technologies.

Interoperability between different grid deployments is a considerable challenge. Field et al. [58] present a comprehensive overview of challenges in grid collaborations, based on their experiences from work on the EGEE project and co-chairing the Grid Interoperation Now (GIN) [136] efforts. The authors describe several approaches to achieve technical interoperability, and conclude that standardization efforts is the best way to achieve technical interoperability despite demonstrating that enforcing standardization is a time consuming and non-trivial task [58]. Field et al. also emphasize the need to not only consider technical difficulties, but also the differences in operational processes which may prevent seamless interoperability [58]. Task metadata management and compatible monitoring are two of the challenges highlighted by Field et al. that are also within the scope of this thesis.

The TeraGyroid project [132] also presents experiences from federated resource usage. In this project they execute tasks on resources belonging to the US TeraGrid [28] and the UK e-Science Grid [80]. They found that they had to port and configure the application to each resource on the grids on which it should be run, and also had to spend considerable efforts to persuade site administrators in both grids to accept certificates issued by the other party [132].

Boghosian et al. [20] provide invaluable insights on the challenges and advantages of grid federations. In this project, the efforts off three different groups are united to create a federated environment to execute applications which are not embarrassingly parallel. Similarly to the TeraGyriod project [132], these groups spent large efforts on interoperability at the user and middleware layers, saying that the "...the probability of success is likely to decrease exponentially with every additional independent grid.". They also state that "Interoperation between Grids today requires much more than just tedious manual effort; it

requires almost heroic effort.", Boghosian et al. found that the primary barrier was not technical, but rather "... the varying levels of evolution and maturity of the constituent Grids." as a result of differences in purposes, priorities, and expertise of the collaborating sites [20].

One of the biggest challenges in federated grids is scheduling [20, 40, 56], especially of non-trivial jobs as the correct execution of a parallel job often means that the job has to be executed in parallel across different sites. The way in which jobs are shared between a set of grids decides the structure and relations of grids within a federation. Fundamental work on distributed scheduling for independent tasks is presented in [106], using meta-schedulers to schedule a common queue of jobs in and between different grids. Other solutions are based on hierarchically organizing grids [17, 83]. Here, a local grid can regard another grid as a very large local resource with special characteristics, and outsource job execution to another grid using standard interfaces.

De Assunção et al. outline the InterGrid [40], a solution based on inter-grid routing analogous to connecting different ISP networks [40, 116], and provides a good overview on the challenges associated with a unified grid. Unfortunately, there are no indications of implementations or practical evaluations of this approach.

Chapter 3

Cloud Computing

Cloud computing has emerged as a broad concept for remote hosting and management of applications, platforms, or server infrastructure, while still offering interactions with remote resources as if they where provisioned locally. The term *cloud computing* originates from the custom of representing computer (or telephone) networks using a drawing of a cloud, hiding the exact location of where things are located or how they are connected. The same analogy applies to computational clouds; the location and other underlying details of remote resources are abstracted and hidden from the user, and the resources are available "on the cloud".

Similarly to grids, cloud computing lacks a crisp and commonly accepted definition and there are many different views (e.g. [64, 68, 75, 176]) as to what constitutes a cloud, and what differs a cloud from a grid (see Section 3.3). Two of the most commonly used definitions originates from the National Institute of Standards and Technology (NIST) [166], and Vaquero et al. [169]. NIST defines [115] cloud computing as:

> *"... a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

This definition is general enough to encompass practically all different cloud approaches, while the one by Vaquero et al. [169] has additional (non-strict) conditions of Service Level Agreements (SLAs) that guarantees capacity to consumers:

> *"Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization.*

This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs."

The above definitions overlap to a large extent, focusing on easy, on-demand access to hardware, application platforms, or services with low delays in the release and provisioning of additional resources. Both definitions employs three widespread service models / scenarios to subdivide the area of cloud computing into subareas:

Infrastructure as a Service (IaaS)
In IaaS solutions, hardware computing resources are made available to consumers as if they were running on dedicated, local machines. The impression of dedicated hardware is commonly achieved by utilizing hardware virtualization techniques, making it possible to host several virtualized system on the same physical host. Some examples of IaaS providers includes Amazon Elastic Compute Cloud (EC2) [7], Rackspace [135], and VMware vCloud Express [172].

Platform as a Service (PaaS)
Instead of offering access to (virtualized) hardware resources, PaaS systems offers deployment of applications or systems designed for a specific platform, such as a programming language or a custom software environment. PaaS systems includes Google App Engine [71], Saleforce's Force.com environment [177], and upcoming projects such as 4Caast [1], CumuloNimbo [37, 131], and Contrail [120], all supported by the European Seventh Framework Programme.

Software as a Service (SaaS)
Web-based applications including, e.g., Microsoft Office Live [117], Google Apps [72] (not to be confused with App Engine), and the gaming platform OnLive [126] are available to consumers online without the need to install and manage the software locally. The software is instead hosted and managed on remote machines, making it possible to run software (including graphic intensive computer games) on remote servers instead of the local machine.

Of these subareas, SaaS and PaaS are normally developed and maintained by a single administrative unit while IaaS sometimes makes use of resources from several different clouds (similarly to federation of grids). Therefore, the remainder of this thesis focuses on IaaS concepts of clouds, and more specifically on the implications imposed by considering and utilizing resources from more than one infrastructure provider. However, many of the managerial concepts described in Chapter 4 can be applied to, e.g., PaaS and SaaS environments as well.

3.1 Virtualization

Hardware virtualization techniques [14, 134] provide means of dynamically segmenting the physical hardware, making it possible to run several different *Virtual Machines* (VMs) on the same physical hardware at the same time. Each VM is a self contained unit, including an operating system, and booting a VM is very much like powering on a normal desktop computer. The physical resources are subdivided, managed, and made available to the executing VMs through a *Hypervisor* (also called VM Monitor).

The concept of virtualization dates from the late 1960s but have been largely unused for quite some time, until it gained renewed interest in the late 1990s. The oft cited reason is that the widespread x86 processor technology was cumbersome and impractical to virtualize compared to its predecessors, and also became cheap enough to increase the number of computers instead of focusing on virtualization [95]. The late 1990s saw efficient software-based virtualization of the x86 platform, and hardware support for virtualization in processors was released in the mid 2000s [3, 22].

Virtualization is the underlying packaging and abstraction technology for basically all IaaS clouds, and there are also several initiatives for using virtualization in HPC and grid computing. For example, Keahey et al. [90] suggest using VMs in grids to, e.g, better meet quality of service demands and provide easier portability between execution environments. Haizea [151] is a scheduling framework utilizing VMs as a tool to maximize utilization while still supporting advance reservations by suspending and resuming VMs. This way,small gaps between jobs can be utilized by resuming a previously suspended VM. An analysis and comparison of virtualization technologies for HPC is presented by Walters et al. [174].

There are several different technologies for virtualization, which Walters et al. [174] present and organize into four different categories:

Full Virtualization Uses a hypervisor to fully emulate system hardware, making it possible to run unmodified guest operating systems at the expense of performance. Well known implementations include VirtualBox [175], Parallels Desktop [130], and Microsoft Virtual PC [81].

Native Virtualization Native virtualization makes use of hardware support in processors to make the costly translations of instructions from full virtualization in hardware instead of software. Known technologies includes KVM [95], Xen [14], and VMware [171].

Paravirtualization In Paravirtualization [178], the operating system in the virtual machine [147] is modified to make use of an API provided by the hypervisor to achieve better performance than full virtualization. Xen [14] and VMware [171] are two well established technologies supporting Paravirtualization.

Operating System-level Virtualization Unix based virtualization systems such as OpenVZ [128] can provide operating system-level virtualization without hypervisors by running several user instances sharing a single kernel.

Virtualization techniques in different categories are generally incompatible, and for paravirtualization there might be interoperability issues even between different versions of the same hypervisor technology. The hardware support makes native virtualization perform almost at the same level as paravirtualization, keeping the losses imposed by virtualization at a couple of percent [3, 14].

There are several benefits of using virtualization in system management (see, e.g., [142]), but the most important ones in the context of this thesis are: VMs are self contained systems, making it possible to execute the VM on all compatible hypervisors; VMs can be paused and resumed; and VMs can be migrated (moved) either by pausing them and resuming them on another host or by moving them without suspending them. Migration a VM without (non-neglectable) downtime is known as *live migration* [32]. There are several schemes for optimizing the migration process, and live migration of VMs can be done with marginal downtime [23, 32, 158]. Being able to execute VMs on remote hosts without severe software dependencies, and the ability to relocate VMs without major effort or downtime forms the core of multi-site cloud computing concept.

3.2 Cloud as an Infrastructure

The starting point of cloud computing as an infrastructure is arguably Amazon [7] offering the provisioning of their resources to anyone, without the need of any application process or long-term commitments, and charging users only for the resources they actually consume.

The quick provisioning of resources makes it possible for consumers to adapt their current resource requirements with very short delays by starting up or stopping VMs according to their needs. To avoid having to customize large amounts of VMs individually, a VM *template* (or type) is often used to start up several identical *instances*[1].

When starting several instances of VMs it is the responsibility of the software running inside each VM to synchronize with the other running instances, for example by registering with a load balancer. Some configuration settings, such as the IP of the load balancer, cannot be encoded into the template itself, either because it is not available until run time or because it needs to be unique for each VM instance. The process of configuring each instance automatically is called *contextualization* [90, 91, 165]. Contextualization is usually performed just prior to booting a VM, and pausing or resuming (or migrating) a VM does not cause another round of contextualization.

[1] These terms are not to be confused with "Instance Types", which are predefined hardware configurations of VMs offered by, e.g., Amazon EC2 [7].

There are three main actors involved in cloud infrastructures, illustrated in Figure 1. The *Infrastructure Provider* (IP) owns and manages the physical resources and any supporting software that is required for infrastructure management. The *Service Provider* (SP) is responsible for the contents the service itself, installing and managing the software running inside the VMs. End users are the consumers of the service offered by the SP.

Even though the actors are conceptually separate, the same organization may of course both own the infrastructure, host services on the infrastructure, and be the end users of their own service. There is also a many-to-many relation between the SPs and IPs, and a single IP normally hosts services from many SPs in a multitennant manner (using the isolation of VMs to keep them from interfering each other). Similarly, a single SP may run services (or even parts of services) on several IPs.

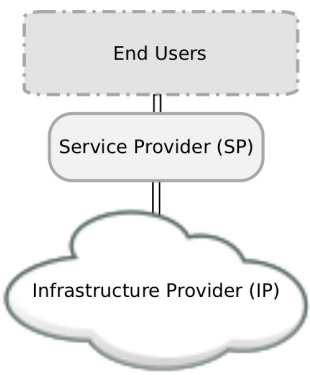


Figure 1: Three main actors for cloud IaaS: the Infrastructure Provider(s) make resources available to Service Provider(s), who in turn offer a software service to End Users.

The IaaS service model is normally offered by the IP, but may have supporting functionality running in the SP. The software running inside the service managed by the SP may consist any type of software, which may (or may not) be other flexible platforms such as PaaS or SaaS solutions. Notably, PaaS or SaaS systems are not required to be hosted on underlying IaaS infrastructures by the service providers, but the variation in resource requirements of PaaS or SaaS systems lends itself well to such solutions. Similarly, SaaS systems may (or may not) be hosted with the support of an underlying PaaS system.

From a resource management perspective, deploying a service to an IP is very much like starting a normal computer application – its lifetime and usage patterns are unknown to the underlying operating system, but the system is still responsible for managing and multitasking different applications without detailed instructions from the user. In an operating system, less prioritized tasks are often neglected in favor of higher prioritized ones, mitigating the problem of insufficient available resources. Similarly, some cloud vendors makes use of

less prioritized instances (such as Amazon's Spot Instances [6]) to increase the utilization when the system is not under heavy load. When resources are running low, the IP can may either free up resources by stopping less prioritized services, or by outsourcing the executions of some VMs to other IPs (see Section 3.4).

Security and privacy concerns are commonly seen as the main limiting factor of clouds as a general utility [64, 85]. Compared to grids, where access usually is preceded by face-to-face identity validations and certificate generation, clouds has a relaxed security model reminiscent of regular Internet sites, using Web based forms for sign up and management, and emails for password retrievals [64]. This relaxed security is a great benefit in terms of usability, but limits the trust of major companies considering using clouds for business-sensitive applications. While the ongoing work on cloud security is progressing (see e.g., [31, 85]), privately hosted and managed clouds has become an option for dealing with sensitive data while still gaining some benefits from the cloud computing paradigm.

Early results of scientific computing using clouds are presented in [89], although most of the results are based on "clouds" where a user has to apply by email for the free execution of a VM during short period of time (hours). The lack of quick on-demand provisioning, the need for manual interactions with the providers prior to execution, and the lack of a utility based business model makes it highly debatable whether the systems used in [89] should be considered clouds at all, or rather an extension of the authors earlier published work on Virtual Workspaces in grids [90].

3.3 Grids and Clouds Compared

While both technologies can be seen as enabling technologies to utilize all kinds of computational infrastructure, the main differences are primarily not about technical solutions; as already mentioned, the utilization of virtualized environments to ease deployment and execution for tasks was known in grids before the cloud era [90]. Instead, clouds and grids have emerged as two different paradigms due to approaching the vision of computing capacity as a utility from different angles.

- Grids are designed to support sharing of pooled resources (normally high performing parallel computers) owned and administered by different organizations, primarily targeting users with hardware requirements surpassing the capacity of commodity hardware (e.g. thousands of processor cores or hundreds of terabytes of storage).

- The development of clouds as a technology is driven by economies of scale [148], where the increased utilization of existing (often commodity) hardware resources offers lower operational expenses for the infrastructure providers, which in turn makes it possible for such providers to offer hardware leasing at prices comparable to in-house hosting.

The differences in scope between the paradigms cause considerable differences in e.g. business models, architecture, and resource management. In the context of this thesis, the most interesting differences are those between grid jobs and cloud services, including how resources are provisioned to the supplied tasks. More in-depth comparisons between clouds and grids can be found in, e.g., [64].

Grid jobs by nature are computational jobs executed on infrastructures with very high (combined) performance, granting exclusive access to resources for the job until it is completed before assigning the resources to the next job in the queue. The capacity requirements and execution time of grid jobs are normally known beforehand, and used as input in job scheduling. Cloud services, on the other hand, are expected to start almost immediately after they are submitted and to run without a fixed execution time until the service is explicitly canceled. The service runs on its assigned share of resources, which may increase or decrease during service execution. Conceptually, the way resources are managed is analogous to time-sharing [137] (grids) vs. space-sharing [157] (clouds) in operating systems.

The extensive use of VMs in cloud computing also means that the delays for starting up and terminating jobs are greater than those of grid computing, as VMs adds quite a bit of overhead in data transfer and start-up times. To generalize, grids are inherently more suitable for applications with high demands on stability and performance by guaranteeing them exclusive access to resources over a short period of time. Clouds are more suitable for less critical long-running tasks suitable for execution on public or shared hardware, and normally offers support for scaling up and down the amount of allocated resources according to the current needs.

The boundaries between grids and clouds are not absolute and generous definitions of either terms creates a large potential overlap. The technologies can also be used in combination. For example, deployment of the Sun Grid Engine (SGE) [69] in a cloud infrastructure is one of the use cases of the RESERVOIR project (see Section 3.4)[141], showing the plausibility of utilizing the flexibility of clouds to host a grid middleware. To make use of the flexibility of the infrastructure, the SGE was deployed using a master VM for job distribution and several instances of worker VMs for job execution, adapting the amount of worker nodes according to the amount of jobs waiting to be executed [33].

Another effort to run cluster software on IaaS infrastructure presented by Keahey et al. is called Sky Computing [92]. In this approach, resources from three different Universities are combined into "Virtual Clusters". Hadoop [179] and Message Passing Interface (MPI) [74] cluster software is hosted on the different VMs, creating a cluster utilizing resources from three university sites.

3.4 Cloud Collaborations

Similarly to federations of grids, clouds can be joined together in different collaboration models to take advantage of the joint infrastructure. While the

main advantage of federated grids is the increased capacity, clouds may also take advantage of collaborations to, e.g., offer geographical redundancy or execute services at geographically advantageous locations otherwise outside the available infrastructure. The economical model of clouds gives rise to several different forms of collaborations, described in Section 3.4.1. In some scenarios, a cloud may provision resources from one or more remote cloud(s) using the regular client interfaces, removing the need for prior resource exchange agreements.

In the basic case, the SP interacts with a single IP and is kept unaware of whether the IP uses resources as a part of a collaboration or not. In collaborative cases, the original IP site where the service was submitted is referred to as the *primary* site, while any collaborating sites are the *remote* sites. The control of the service and responsibility towards the SP remains in the primary site regardless of where the service is actually executed, and the primary site is also responsible for ensuring that SLAs are maintained or compensated for. To be able to utilize remote resources, the use of resources between IP sites may be governed by separate SLAs or *framework agreements* [24], stipulating the terms of resource exchange between IPs.

As with grid computing, the use of several clouds introduces a lot of heterogeneity problems that ultimately only can be resolved using standardization efforts. Native (hardware) virtualization is a first major step to standardization on the lowest hardware level. There are also efforts to create standardized and general formats for specifying virtual machines and virtual hard drives [42, 118] and general cloud APIs [34, 41, 113, 127], but neither standard has yet emerged as a generally accepted candidate.

VM incompatibility issues aside, there are a number of operational challenges imposed by the used of collaborative clouds. Since each site retains its full autonomy, and its own policies and objectives, the internal workings of each site are largely obscured to other sites in the collaborations. This means each site only has details available regarding local resources, and at best incomplete information regarding the state in other sites. Service provisioning across clouds therefore has to be based on probabilities and statistics rather than complete information. Another challenge not present in single clouds is that sites participating in collaborations may have external events affecting the state of a service and resource availability of the infrastructure. For example, a remote site may place services on the infrastructure of the primary site, or force the withdrawal of VMs running on the infrastructure of the remote site and therefore forcing the primary site to re-plan the placement across the infrastructure.

The RESERVOIR (Resources and Services Virtualization without Barriers) [138, 139, 140, 180] project focus on creating and validating the concept of cloud federations across several infrastructure providers through several use cases, including running SGE [69] and SAP [145] applications on the federated infrastructure. One of the results of the project is the design and creation of Virtual Application Networks (VANs) [78]. These overlay networks, extending previous work from e.g. [164], offers one solution to allow VMs being a part of internal private networks to be migrated to other sites in the federation without

being disconnected. These VANs can be used to manage monitoring information for services spanning several cloud sites, see Section 4.1. The RESERVOIR project also outlines a common specification for cloud services [66, 139] to facilitate interoperability, made by extending the standard Open Virtualization Format [42].

The OPTIMIS [57] project targets the creation of a toolkit of components able to (among other things) support multiple cloud scenarios without extensive changes to the software itself. The project [57] also outlines interesting conflicts of interest between the different actors (SPs and IPs). For example, the ambition of the IP to maximize profit is usually contradictory to the SP ambitions of hosting services at a low cost without neglecting the service performance.

3.4.1 Cloud Computing Scenarios

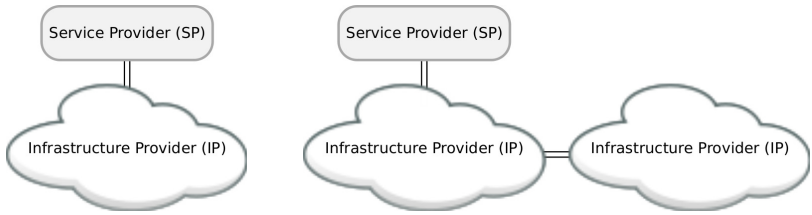
The relation between different clouds in collaboration is commonly modeled as different deployment scenarios [57, 139], depending on the type of interactions between the different sites in the collaboration. We divide the scenarios into three main categories, *federated clouds*, *multi-clouds*, and *private clouds*, each described and illustrated in the coming subsections. Different scenarios can also be combined into *hybrid clouds*, with *bursted private clouds* commonly used as an example. Note that all collaboration scenarios are *multi-clouds* in the sense that they span more than one cloud. The term is used in this more general sense in the title of this thesis, but used in a more specific case in this subsection to describe a specific collaboration scenario. This is done in order to stay in line with, e.g., [57].

Figure 2a shows a simplified model of a standard cloud which is used as the starting point when describing the other deployment scenarios. As previously mentioned in Chapter 3, a single IP normally hosts the services of several SPs, although only a single SP is shown in the illustrations.

Federated Clouds

Federations of clouds (Figure 2b) are formed at the IP level, making it possible for infrastructure providers to make use of remote resources without involving or notifying the SP owning the service. Gaining access to more resources is not the only potential benefit of placing VMs in a remote cloud. Other reasons include fault tolerance, economical incentives, or the ability to meet technical or non-technical constraints (such as geographical location) [138] which would not be possible within the local infrastructure.

Provisioning of remote resources through federations can be done with several remote sites at the same time, using factors such as cost, energy efficiency, and previous performance to decide which resources to use [57]. In some cases, a service may be passed along from a remote site for execution at a third party site, creating a chain of federations. As each participant in the chain is only



(a) A standard cloud deployment.

(b) Two cloud IPs form a federation.

Figure 2: The illustration on the left shows a standard cloud scenario, where one or more SPs are using the resources of a single IP. In the federated case, shown on the right, an IP may employ other IPs to host (parts of) the running services without involving the SP.

aware of the closest collaborating sites, special care has to be taken in the VM management and information flow in such scenarios [53].

Multi-Clouds

The scenario where the SP itself is involved in moving and prioritizing between different IP offerings is called a *Multi-cloud* [57] scenario. In this case, illustrated in Figure 3, the SP is responsible for planning, initiating, and monitoring the execution of services running on different IPs. Any interoperability issues has to be detected and managed by the SP, affecting the set of sites which can be used for multi-cloud deployments.

The automatic selection and management of different alternatives using *brokers* is a well known approach for, e.g., grid computing [56, 93]. As shown in [57, 163], brokers can also be used as an intermediate component in multi-cloud scenarios. In this case, illustrated in Figure 4, the broker is placed between the SP and the IP. The broker may act as an SP to the IP and as an IP to the SP, containing a lot of the complexity of multi-cloud deployments within the broker itself [57].

Tordsson et al. [163] provide an overview and practical experiences of cloud brokering, including quantified results of performance gained from the brokering of resources belonging to different cloud providers.

Private Clouds

Private clouds, shown in Figure 5a are cloud deployments hosted within the domain of an organization or a company not made available for use by the general public [10]. Such deployments circumvents many of the security concerns related to hosting services in public clouds by keeping the execution within the same security domain, while still offering a computational infrastructure to internal users.

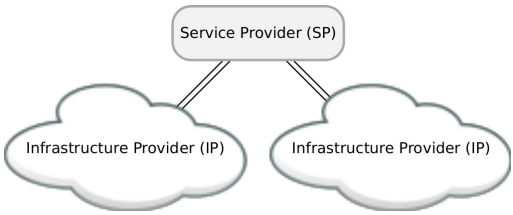


Figure 3: In multi-cloud scenarios, the SP itself may control and decide the deployment of a service using several different IPs.

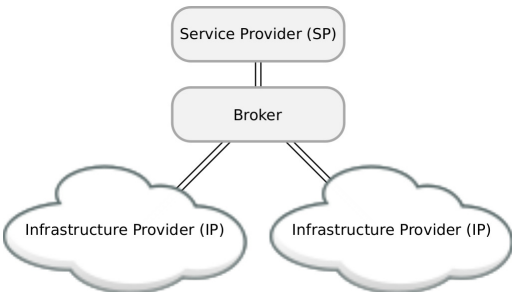


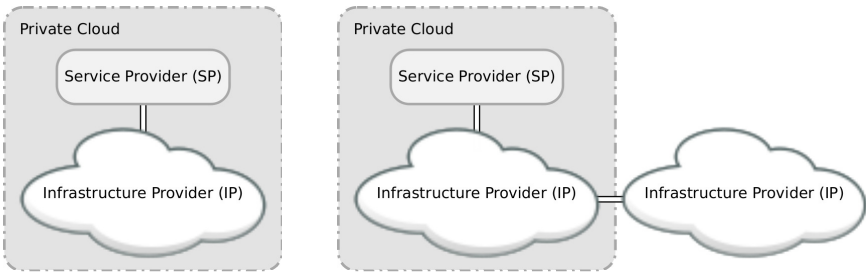
Figure 4: In brokered multi-cloud, a dedicated broker component is used by the SP to simplify the deployment and management process.

Similarly to grids, private clouds only have a finite set of resources and therefore the infrastructure must at some point, prioritize, enqueue, or reject service requests in order to satisfy SLA agreements [153]. It is also likely that private clouds are based on collaboration models between peers rather than pay-per-use alternatives. This creates a need for a service model closer to that of grids than public clouds, and so far there has been little focus in literature on the specific challenges of private clouds.

Hybrid Clouds

Hybrids between different scenarios can be used to overcome limitations of single usage scenarios. For example, to avoid the problem of finite resources in private clouds, such clouds may temporarily employ the resources of external public cloud providers. These *bursted private clouds* (described in e.g., [153]) offers a combination of the security and control advantages of private clouds and the seemingly endless scalability of public clouds, but requires very sophisticated placement policies to guarantee the integrity of the system. The relation between private and bursted private clouds is illustrated in Figure 5.

Sotomayor et al. [153] outline the general concepts of hybrid clouds and provides an overview of different cloud technologies and their support for hybrid models. In their work, OpenNebula [152] is used to create hybrid cloud solutions



(a) Private cloud. (b) Bursted private cloud (hybrid)

Figure 5: Private clouds offer stronger guarantees on control and security as the whole infrastructure can be administered within the same security domain. If needed, private clouds may have less sensitive tasks be executed on a public cloud instead, forming a hybrid cloud scenario commonly referred to as bursted private cloud.

based on a private infrastructure and a set of cloud drivers used to burst to different external providers such as Amazon EC2 [7] or ElasticHosts [46].

Chapter 4

Task Metadata Management

The primary focus of this thesis is the collection, management, and use of task metadata in distributed and multi-provider infrastructures such as grids and clouds. Previous chapters have introduced the fundamental concepts of the main paradigms, including different collaboration models, and this chapter outlines internal infrastructure procedures related to task metadata.

The task metadata contains information about, e.g., the duration, status, and resource consumption of a running task, and forms the primary source of feedback for different internal procedures in the infrastructure. The following sections covers gathering and managing of task metadata, and describes different internal grid and cloud infrastructure processes using the metadata as the primary input.

4.1 Monitoring

Monitoring is the process of gathering information about infrastructure or a service during run time. In grid systems, the focus of monitoring lies on the health, performance, and status of the infrastructure resources [173, 183]. This information is subsequently used for fault detection and recovery, prediction of resource performance, and also to tune the system for better performance [162]. Grid monitoring is slightly out of scope regarding task metadata management, as monitoring is normally not performed regarding the grid jobs themselves (see [183] for a comprehensive overview of grid monitoring). Instead, metadata concerning the result and status of a grid job is collected once the job has terminated (in the shape of usage records), regardless of if the job succeeded to complete successfully or not. Creation and management of these records are further discussed in Section 4.2.

Monitoring of running services is fundamental in clouds as monitoring data

is the primary input used in most internal management procedures. The lack of compatible monitoring is one of the main incompatibility hurdles of cross-site clouds [10, 103]. There are three different kinds of monitoring data used in clouds, measurements from the infrastructure, the hypervisor, or from within the service itself:

- Infrastructure specific measurements showing the health and utilization of physical resources. Monitoring the state of infrastructure resources is not a specific problem for cloud computing, and the same tools used for general purpose system monitoring (such as Nagios [16], Ganglia [114], or collectd [59]) can be used also in these contexts.

- Data concerning the resource consumption of individual VMs running on the hardware can be obtained by communicating with the VM hypervisor, or by using tools (such as the libvirt [109] API) that are capable of operating across several different hypervisors. The VM information is commonly used to perform the fulfillments of SLAs or as input to elasticity and service profiling.

- Service specific Key Performance Indicators (KPIs) are used to measure and manage monitoring values specific to the service. These values are normally only available from inside the service software itself, and might constitute values such as the current number of active sessions to a Web based application or the number of concurrent transactions in a database system. These values can be used to perform, e.g., elasticity.

Measuring and managing monitoring KPIs from inside the service itself is an interesting problem that is not yet well studied [86]. Some cloud solutions (such as RESERVOIR [139]) have a strong separation between service management and the VM itself, in the sense that the VM is unaware of the location of the management components, and the management components are unaware of the location of the VM. This *location unawareness* [48, 53, 78] has a great influence on what techniques can be used to make the service specific data available to the cloud infrastructure from inside the VMs.

An important factor to consider in cloud collaborations is that more than one site might be interested in the monitoring data produced for a given service. For this reason, naive solutions such as sending the data from inside the VM to an external internet endpoint cannot be used in, e.g., federation scenarios, as the data would not be visible to the infrastructure on the remote site[1]. There is also no guarantee that all VMs of a service has external network access [78]. Instead, the monitoring data has to flow back from the executing site to the primary site through any intermediaries (if any).

The Lattice framework [33] presents a solution for service level monitoring based on customized virtual networks (VANs [78]) to pass measurements from inside the VMs to the infrastructure on the outside without external network

[1] Recall that VMs are not re-contextualized when they are migrated.

access. In this solution, the functionality of the network broadcast directive is overridden and used for monitoring tasks instead. However, without the customized virtual networks this solution would not be possible, and so this is not a generally applicable alternative.

An alternative based on File System in User Space (FUSE) [160] is outlined in [53]. In this solution, FUSE is used to create a small application that simulates a hard drive partition. File system calls (such as writes) result in a normal programmer controlled method call in the application, and the complexity of externalizing the data can be hidden inside the FUSE based application. The problems of actually externalizing the data without knowing the location of some management component remains unsolved, however.

An architecture and implementation of a service oriented monitoring framework for use in cloud infrastructures is presented in [87]. This approach does not seem to consider the problem imposed by service level management nor federations, but instead focuses on monitoring of information from different sources and for use in real-time applications.

4.2 Accounting and Billing

Accounting systems are responsible for metering and managing records on resource consumption by users in grids or clouds. In grids, a *Usage Record* [112] for a job is usually created once the job has finished executing. The usage record contains a lot of general metadata about the job, such as when it was started and finished, and may also contain a summary of the combined resource consumption of a job in terms of, e.g., amount of data transferred on the network. Cloud systems normally rely on run time monitoring of service resource consumption as a basis for accounting.

In federations of grids, the accounting data generated upon job completion is usually important both for the originating grid site, the executing grid site, and possibly any consortium or organization linking these resources together. Managing usage records in such environments is the subject of Paper I [52]. For cross-site cloud computing, the aggregation of data from different site is usually managed by the underlying monitoring system, as accounting is not the only internal cloud process depending on the aggregated raw monitoring data.

One of the major differences between grids and clouds is the underlying economical model, which can be clearly linked to the origins of each paradigm and to the niches they occupy today. For grids, the most common solutions are based collaborative sharing models where the usage data is converted to abstract currencies [15, 67, 133]. Abstract credits are normally awarded to users through an out-of-band application procedure, in which a steering committee allocates credits to different projects based on scientific merit. These credits can then be exchanged for computing time on the infrastructure. There are numerous suggestions on how to achieve economical models and architectures for use in grids, commonly based on auctions or other market-based schemes,

some of which can be found in [12, 25, 27, 50, 101, 182]. Nakai and Van Der Wijngaart [122] presents an in-depth economical analysis of the feasibility and expectations of markets in grid scheduling, proving that the use of markets is not generally applicable and may not lead to the desired outcomes [122].

Many grid accounting systems also support converting the abstract currencies to real monetary units (at least by easily extending the core mechanisms), but real economical models for grid usage has never been widely adopted. One reason could be that the allocation of abstract credits means that stakeholders can partly affect the utilization of the infrastructure. The use of real money could mean that smaller projects could be constantly outbid by other consumers, preventing them from utilizing the common infrastructure.

In public clouds, users are free to request as much resources as they require on the short term, and paying only for the resources they are currently requesting. In such systems, the accounting data (based on monitoring) is used as input in the *billing* process, converting the hardware measurements to real monetary bills using different pricing schemes.

The two major payment models used in clouds are *prepaid* and *postpaid*, used in the same manner as in the mobile-phone industry. Prepaid, where credits are purchased in advance and consumed in accordance with resource consumption, offers greater control over the maximum costs but running out of credits may cause the service to stop executing. Postpaid, where the consumer is billed at regular intervals for the previous usage, is more sensitive to unexpected amounts of resource consumption, but does not risk running out and hence disturbing the service execution.

Many cloud providers employ overbooking strategies [143] and sell more resources than is actually available, relying on probabilistic models that not all resources are be requested at the same time [76]. However, overbooking strategies ultimately leads to increased amounts of broken SLAs, and each broken SLA generates compensations to the SP. Therefore, dealing with both costs and compensations is a major requirements for accounting in clouds. Birkenheuer et al. [19] show that overbooking schemes are valid options and can achieve a 20% increase in profit even when considering compensations for broken SLAs.

Deployment scenarios such as bursted private clouds or cloud federations offers seemingly unlimited hardware resources, as there may always be resources available at collaborating sites. In theory, this means that also the amount of accounting (and monitoring) data generated by services in the cloud is unlimited. Accounting data is commonly considered financial data, with means there are high demands on storing and managing such data over a long period of time (at least ten years in some jurisdictions). This creates a resource provisioning problem for the management of accounting data similar to the problem addressed by cloud computing itself. Totally scalable solutions for such data has not yet been fully established, but initial work on this subject can be found in, e.g., [39, 110].

4.3 Scheduling and Placement

The process of assigning incoming tasks to available resources, usually denoted *Scheduling* for grids and *Placement* for clouds (although scheduling is sometimes used also for cloud services), is one of the internal processes often relying on task metadata for future decisions.

In grid computing, fairshare scheduling [38, 45, 49] is a wide-spread approach where the scheduler tries to distribute computational resources according to predefined usage shares. The scheduler normally operate on aggregated task metadata for each user and compares the previous usage to the users predefined allocation of resources, using the difference between promised and utilized resources as a factor for prioritizing incoming jobs. The data used in the fairshare process is usually based on usage records, obtained either by querying the underlying accounting system or by receiving such records straight from the infrastructure. The accuracy and availability of usage records and the delay before the data is made available to the fairshare scheduler directly affects the performance and convergence time in the system. Preliminary results in quantifying the relation between task metadata management and job convergence is presented in Paper IV, and further evaluation of these factors are part of future work.

Similarly to grid scheduling, cloud placement can be focused on several objectives and the objectives of each autonomous site may be different [24]. Even within a single site there might be several conditions to consider, and there is often a trade-off between multiple factors such as maximizing the utilization of the infrastructure while minimizing the risk of breaking SLAs. Currently, the amount of broken SLAs seems to be the primary means of measuring the suitability of a cloud deployment. The placement problem takes very different forms in different cloud scenarios. The limited resources in private clouds creates a need for similar solutions as employed in grid computing, as the total amount of requested capacity will be larger than the available resources at some point [153]. In public clouds, the resources are seemingly unlimited and solutions of the placement problem for an IP can focus on optimizing the revenue while minimizing the risk of breaking SLAs [24]. Hybrid scenarios such as bursted private clouds have different challenges as the utilization of the limited local resources must be balanced with the higher costs (and insecurity) of the public resources. As shown by Van den Bossche et al. [168], approaches which perform very well in public clouds may perform drastically worse in bursted private cloud settings due to very large differences in the required time to find an optimal solution when considering also the internal resources. Similarly to fairshare scheduling in grids, the quality and availability of monitoring data and the delays imposed by collaborations is likely to have a great impact on cloud placement, but quantification and further analysis of these areas are subject to future work.

4.4 Elasticity

The ability to quickly request or release resources in response to the current load of a service is one of the most prominent features of cloud computing. *Elasticity* is the process of automating the decisions for when to scale up or down and transfer the decision making from human administrators to processes running in SP or in the infrastructure. By specifying a set of *Elasticity Rules* [141] and include the rules in the service manifest [66], the rules for scaling a service becomes an integral part of the service itself. The rules can be used to specify, e.g., how many users can be served by each VM instance, which may be used in combination with reactive or predictive models to calculate the number of required instances [4].

There are two types of elasticity, *horizontal elasticity* and *vertical elasticity* [4]. In horizontal elasticity, the number of VM instances of a certain type is increased or decreased to correlate with the current load. In vertical elasticity, the amount of hardware resources assigned to one or more VM(s) (such as the amount of RAM or number of CPUs) is dynamically increased or decreased. Horizontal elasticity puts additional strain on the application running inside the VMs, as the system itself must synchronize the tasks between the different instances. Vertical elasticity, on the other hand, requires that the operating system and application running inside the VM is capable of making efficient use of, e.g., a dynamic amount of available RAM.

The elasticity process is normally based on monitoring data concerning the hardware consumption of the VM, or on KPIs monitored from inside the application itself. To shorten the reaction time, elasticity requires up do date measurements regarding the state or KPIs of each VM regardless of where in the (cross-site) infrastructure each VM is running. Normally, the time required for instantiation of new VMs is a few minutes, but recent efforts by, e.g., Lagar-Cavilla et al. [100] has shown that new VMs can be started up in the matter of seconds using techniques similar to *fork* system calls. To avoid becoming the bottleneck, performance is a key requirement for monitoring solutions designed to support rapid elasticity.

Chapter 5

Summary of the Papers

The publications in this thesis focus on different aspects of task data management, including collecting of task data, management of the data within a distributed grid or cloud environment, and how the collected data can be used in different internal procedures such as accounting, billing, or job scheduling.

5.1 Paper I

Paper I [52] investigates how task data collected across autonomous nodes in different distributed grid or cloud usage scenarios can be managed and shared to other parties in the collaboration. The scenarios considered in this paper are hierarchies of grids, mutual grid collaborations, and federations of clouds.

In the paper, we identify a set of requirements from the different usage scenarios, and use these requirements to evaluate several different approached to task data sharing in these environments. This process results in the implementation and evaluation of a light-weight component (LUTSfed) controlling the flow of usage information between different parts of the collaboration. This process is made non-intrusive and optional by reusing existing read and write interfaces of the data management components, and adds support for different cardinality (one-to-many, many-to-one, many-to-many) in usage sharing. We demonstrate how the LUTSfed component can be used to realize the three different usage scenarios by configuring and deploying the component in different ways, without affecting the operation of the already running data management components in different parts of the collaboration.

The performance of the stand-alone LUTSfed component is evaluated in different scenarios, including relations of different cardinality between the number of source and target components and the performance losses inferred by using the LUTSfed component.

5.2 Paper II

Paper II [48] investigates accounting and billing in the federated cloud environments introduced in the RESERVOIR project. The paper is based on two new use cases not present in traditional grid and cloud environments; accounting for cloud services for which the number of sub-components is dynamic and unknown to the accounting system, and accounting for cloud services in which also the placement of sub-components in the federated infrastructure is dynamic and unknown.

A set of requirements for an accounting and billing system in federated clouds is formulated based on the use cases and general non-functional requirements. Existing grid accounting systems are evaluated based on these requirements, but no existing alternative is found to fully support the set of requirements imposed by this environment. Instead, a new architecture for a cloud focused accounting and billing subsystem is proposed. This new architecture is designed from the start to fulfill the requirements imposed by the federated cloud environment. Paper II describes the proposed architecture is detail.

5.3 Paper III

Paper III [103] provides a unified view on a set of managerial challenges present in cloud federations, namely representation, placement, and monitoring of cloud services. A cloud service may constitute of several different subsystem, such as internal networks and shared file storage nodes, and may also have different requirements and restrictions for placement in different parts of the service. Paper III presents a model for expressing the internal structure of cloud services including, e.g., geographical or intra-component affinities (placement restrictions).

When performing placement of cloud services, the placement restrictions expressed in the service structure must be considered. Migrating (moving) parts of the service to another host may require cascading migrations in order to adhere to the specified affinities. We present a model for placement that abides the specified constraints, and extend this model with a heuristic to determine which parts of the service that are suitable for migration.

The placement process is partly based on monitoring data collected from different parts of the cloud federation. The paper presents a data distribution architecture based on semantic metadata annotations that may be used to bridge the gap imposed by monitoring systems in different parts of the cloud federations.

5.4 Paper IV

Paper IV [129] presents the design and functional evaluation of a grid-wide support system for job prioritization based on fairshare allocations, based on

earlier work published in [49].

The proposed system is a distributed, stand-alone, support-system usable by job schedulers to externalize the fairshare prioritization procedure. The paper presents a distributed tree-based policy model for specifying user shares hierarchically, making it possible for a project to subdivide its own share of usage into specific shares per user and/or sub-project. The paper describes an algorithm for prioritizing user jobs based on predefined user shares and historical usage data. Finally, the decentralized architecture used to realize the system is described in detail. The overall system behavior and its ability to accurately prioritize jobs in different scenarios is demonstrated, showing that system is capable of achieving grid-wide fairshare also in the presence of dynamically changing policies and run-time site failures. Performance results show that the convergence rate of the system is greatly affected by delays in data updates, highlighting the relation between task metadata management and system performance.

Chapter 6

Future Work

As discussed in this thesis, task metadata management is a fundamental task in both grids and clouds and the data is used as the primary input to many different processes. The following sections present categories of future work related to the topics described in the thesis.

6.1 Service Monitoring

As outlined in Section 4.1, the problem of monitoring of internal service data is an open but important problem in cloud computing. The data extracted from the application running in the VM is commonly used in, e.g., accounting and elasticity, and data extraction must be coherent and with low delays regardless of if the site is participating in a collaboration or not. A general solution for extracting information from inside the VM and making it available to the cloud infrastructure is one potential area for future work, possibly considering the aid of the hypervisor software itself by employing specific system calls similar to those used in paravirtualization.

6.2 Accounting and Billing

Research on accounting and billing is so far focused on IaaS clouds, quantifying resource consumption in similar ways as in grid computing. A possible future work direction is to investigate if and how these systems would have to evolve to be applicable also to PaaS and SaaS environments.

Private (and hybrid) clouds are popular alternatives for hosting sensitive applications, but so far accounting and billing for these kinds of clouds have not been thoroughly explored. The usage models of private clouds is most likely closer to those of a collaborative grid than a public cloud, and the monetary based compensation system used in public clouds may therefore not be applicable. Similarly, bursted private clouds probably has to incorporate

limitations on how much external resources may be provisioned, and by which users [168].

As briefly mentioned in Section 4.2, the amount of accounting and billing data in scenarios such as bursted private clouds or cloud federation can potentially be unlimited. Since this data is commonly required to be stored and managed for a long period of time provisioning resources for management of accounting and billing data is a resource scaling problem yet unsolved. This problem is briefly discussed in [110], outlining record data aggregation and scalable database back-ends as two possible approaches to this problem. However, further work is required to determine the implications on data consistency and durability if using scalable database back-ends such as ElasTras [39], Cassandra [102] or BigTable [29].

6.3 Fairshare Scheduling

Future work on fairshare scheduling in grids as outlined in Paper IV [129] includes more in-depth analysis of different algorithms for calculating fairness based on the historical usage and user allocations. Different algorithms and different settings of parameters such as the amount of historical data to consider is likely to have a large impact on the behavior of the system.

Early results on the impact of task metadata management on the accuracy of fairshare is presented in Paper IV, and further analysis and quantification within this area is subject to future work. As outlined in [49] the inclusion of estimated times for running jobs in the fairshare process may also have a great impact on the accuracy and convergence rate of the system, and is one possible avenue for further studies especially with regard to the additional requirements on metadata management associated with dealing also with running jobs.

Further integration work with different scheduler software and evaluation of the system performance over a long time in a real deployment is also part of future work.

Bibliography

[1] 4CaaSt project. Morfeo 4CaaSt. `http://4caast.morfeo-project.org/`, September 2011.

[2] J. Abbate. From ARPANET to INTERNET: A history of ARPA-sponsored computer networks, 1966-1988. 1994.

[3] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 2–13. ACM, 2006.

[4] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An Adaptive Hybrid Elasticity Controller for Cloud Infrastructures. 2011. Submitted.

[5] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. Van Nieuwpoort, A. Reinefeld, F. Schintke, et al. The grid application toolkit: toward generic and easy application programming interfaces for the grid. *Proceedings of the IEEE*, 93(3):534–550, 2005.

[6] Amazon.com, Inc. Amazon EC2 Spot Instances. `http://aws.amazon.com/ec2/spot-instances/`, September 2011.

[7] Amazon.com, Inc. Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2`, September 2011.

[8] D. Anderson. BOINC: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, 2004.

[9] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, A. S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) specification, version 1.0. `http://www.ogf.org/documents/GFD.136.pdf`, September 2011.

[10] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[11] H. Bal, C. de Laat, S. Haridi, K. Jeffery, J. Labarta, D. Laforenza, P. Maccallum, J. Mass, L. Matyska, T. Priol, et al. Next Generation Grid (s) European Grid Research 2005–2010. *Information Society Technologies, European Commission, Expert Group Rep*, 2003.

[12] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-based load management in federated distributed systems. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation-Volume 1*, pages 15–15. USENIX Association, 2004.

[13] J. Baldassari, D. Finkel, and D. Toth. SLINC: A Framework for Volunteer Computing. In S. Zheng, editor, *Proceedings of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems*, 2006.

[14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177. ACM, October 2003.

[15] A. Barmouta and R. Buyya. GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration. In *Workshop on Internet Computing and E-Commerce, Proceedings of the 17th Annual International Parallel and Distributed Processing Symposium (IPDPS 2003), IEEE Computer Society Press, USA, April*, pages 22–26, 2003.

[16] W. Barth. *Nagios: System and Network Monitoring*. No Starch Press, San Francisco, CA, USA, 2nd edition, 2008.

[17] C. Baumbauer, S. Goasguen, and S. Martin. Bouncer: A globus job forwarder. In *Proc. 1st TeraGrid Conf*, 2006.

[18] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, et al. Adaptive computing on the grid using AppLeS. *Parallel and Distributed Systems, IEEE Transactions on*, 14(4):369–382, 2003.

[19] G. Birkenheuer, A. Brinkmann, and H. Karl. The gain of overbooking. In *Job Scheduling Strategies for Parallel Processing*, pages 80–100. Springer, 2009.

[20] B. Boghosian, P. Coveney, S. Dong, L. Finn, S. Jha, G. Karniadakis, and N. Karonis. Nektar, spice and vortonics: Using federated grids for large scale scientific applications. In *Challenges of Large Applications in Distributed Environments, 2006 IEEE*, pages 34–42. IEEE, 2006.

[21] M. Bote-Lorenzo, Y. Dimitriadis, and E. Gómez-Sánchez. Grid characteristics and uses: a grid definition. In *Grid Computing*, pages 291–298. Springer, 2004.

[22] J. S. Bozman and G. P. Chen. Optimizing Hardware for x86 Server Virtualization. White Paper.

[23] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pages 169–179. ACM, June 2007.

[24] D. Breitgand, A. Marashini, and J. Tordsson. Policy-Driven Service Placement Optimization in Federated Clouds. Technical Report H-0299, IBM Research Report, 2011.

[25] J. Brunelle, P. Hurst, J. Huth, L. Kang, C. Ng, D. Parkes, M. Seltzer, J. Shank, and S. Youssef. Egg: An extensible and economics-inspired open grid computing platform, 2006.

[26] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In *hpc*, page 283. Published by the IEEE Computer Society, 2000.

[27] R. Buyya, D. Abramson, and S. Venugopal. The grid economy. *Proceedings of the IEEE*, 93(3):698–714, 2005.

[28] C. Catlett. The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility. In *CCGrid*, page 8. Published by the IEEE Computer Society, 2002.

[29] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*, 2006.

[30] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. *W3C working draft*, 26, 2004.

[31] M. Christodorescu, R. Sailer, D. Schales, D. Sgandurra, and D. Zamboni. Cloud security is not (just) virtualization security: a short paper. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 97–102. ACM, 2009.

[32] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 273–286. ACM, May 2005.

[33] S. Clayman, A. Galis, C. Chapman, G. Toffetti, L. Rodero-Merino, L. Vaquero, K. Nagin, and B. Rochwerger. Monitoring Service Clouds in the Future Internet. In *Towards the Future Internet - Emerging Trends from European Research*, pages 115–126, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

[34] Cloud Computing Interoperability Forum. Unified Cloud Interface Project. `http://www.cloudforum.org/`, September 2011.

[35] D. Comer. The computer science research network CSNET: A history and status report. *Communications of the ACM*, 26(10):747–753, 1983.

[36] CoreGRID. CoreGRID annual report 2007. `http://www.coregrid.net/mambo/content/view/310/301/`, September 2011.

[37] CumuloNimbo project team. CumuloNimbo: Highly Scalable Transactional Multi-Tier PaaS Main menu. `http://www.cumulonimbo.eu/`, July 2011.

[38] E. Dafouli, P. Kokkinos, and E. Varvarigos. Fair Execution Time Estimation Scheduling in Computational Grids. *Distributed and Parallel Systems*, pages 93–104, 2008.

[39] S. Das, S. Agarwal, D. Agrawal, and A. El Abbadi. Elastras: An elastic, scalable, and self managing transactional database for the cloud. 2009.

[40] M. De Assunção, R. Buyya, and S. Venugopal. InterGrid: A case for internetworking islands of Grids. *Concurrency and Computation: Practice and Experience (CCPE)*, 20(8):997–1024, 2008.

[41] Distributed Management Task Force. Cloud Management Standards. `http://www.dmtf.org/standards/cloud`, September 2011.

[42] Distributed Management Task Force, Inc. Open Virtualization Format Specification. DMTF 0243 (Standard), Feb. 2009.

[43] N. Doulamis, E. Varvarigos, and T. Varvarigou. Fair Scheduling Algorithms in Grids. *IEEE Transactions on Parallel and Distributed Systems*, 18:1630–1648, 2007.

[44] K. Dowd. *High performance computing*. O'Reilly & Associates, Inc., 1993.

[45] C. L. Dumitrescu, M. Wilde, and I. Foster. A model for usage policy-based resource allocation in grids. *Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on*, pages 191 – 200, June 2005.

[46] ElasticHosts Ltd. ElasticHosts. `http://www.elastichosts.com/`, September 2011.

[47] M. Ellert, M. Grønager, A. Konstantinov, B. Konya, J. Lindemann, I. Livenson, J. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced Resource Connector middleware for lightweight computational Grids. *Future Generation computer systems*, 23(2):219–240, 2007.

[48] E. Elmroth, F. Galán, D. Henriksson, and D. Perales. Accounting and Billing for Federated Cloud Infrastructures. In *GCC '09: Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing*, pages 268–275, Washington, DC, USA, 2009. IEEE Computer Society.

[49] E. Elmroth and P. Gardfjäll. Design and evaluation of a decentralized system for Grid-wide fairshare scheduling. In H. Stockinger et al., editors, *First International Conference on e-Science and Grid Computing*, pages 221–229. IEEE CS Press, 2005.

[50] E. Elmroth, P. Gardfjäll, O. Mulmo, and T. Sandholm. An OGSA-Based Bank Service for Grid Accounting Systems. In J. Dongarra et al., editors, *Applied Parallel Computing. State-of-the-art in Scientific Computing*, volume 3732 of *Lecture Notes in Computer Science*, pages 1051–1060. Springer-Verlag, 2005.

[51] E. Elmroth, P. Gardfjäll, A. Norberg, J. Tordsson, and P.-O. Östberg. Designing general, composable, and middleware-independent Grid infrastructure tools for multi-tiered job management. In T. Priol and M. Vaneschi, editors, *Towards Next Generation Grids*, pages 175–184. Springer-Verlag, 2007.

[52] E. Elmroth and D. Henriksson. Distributed Usage Logging for Federated Grids. *Future Generations Computer Systems*, 26(8):1215–1225, 2010.

[53] E. Elmroth and L. Larsson. Interfaces for Placement, Migration, and Monitoring of Virtual Machines in Federated Clouds. In *Eighth International Conference on Grid and Cooperative Computing (GCC 2009)*, pages 253–260, Los Alamitos, CA, USA, August 2009. IEEE Computer Society.

[54] E. Elmroth and J. Tordsson. A Grid resource broker supporting advance reservations and benchmark-based resource selection. In J. Dongarra, K. Madsen, and J. Waśniewski, editors, *Applied Parallel Computing - State of the Art in Scientific Computing, Lecture Notes in Computer Science vol. 3732*, pages 1061–1070. Springer-Verlag, 2006.

[55] E. Elmroth and J. Tordsson. Grid Resource Brokering Algorithms Enabling Advance Reservations and Resource Selection Based on Performance Predictions. *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, 24(6):585–593, 2008.

[56] E. Elmroth and J. Tordsson. A standards-based Grid resource brokering service supporting advance reservations, coallocation and cross-Grid interoperability. *Concurrency Computat.: Pract. Exper.*, 21(18):2298–2335, 2009.

[57] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan. OPTIMIS: a holistic approach to cloud service provisioning. 2011. Accepted.

[58] L. Field, E. Laure, and M. Schulz. Grid deployment experiences: Grid interoperation. *Journal of Grid Computing*, 7(3):287–296, 2009.

[59] Florian Forster. collectd. `http://collectd.org/`, September 2011.

[60] I. Foster. What is the grid? a three point checklist. *GRID today*, 1(6):32–36, 2002.

[61] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computer Science and Technology*, 21(4):513–520, 2006.

[62] I. Foster and C. Kesselman. *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann, 2004.

[63] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[64] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[65] P. Fowler, S. Jha, and P. Coveney. Grid-based steered thermodynamic integration accelerates the calculation of binding free energies. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 363(1833):1999, 2005.

[66] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Vaquero. Service Specification in Cloud Environments Based on Extensions to Open Standards. In *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, COMSWARE '09, pages 19:1–19:12, New York, NY, USA, 2009. ACM.

[67] P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, and T. Sandholm. Scalable Grid-wide capacity allocation with the SweGrid Accounting System (SGAS). *Concurrency Computat.: Pract. Exper.*, 20(18):2089–2122, 2008.

[68] J. Geelan. Twenty One Experts Define Cloud Computing. *Virtualization*, August 2008. Electronic Magazine, article available at `http://virtualization.sys-con.com/node/612375`.

[69] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 35–36. IEEE, 2001.

[70] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. Von Laszewski, C. Lee, A. Merzky, H. Rajic, and J. Shalf. SAGA: A Simple API for Grid Applications. High-level application programming on the Grid. *Computational Methods in Science and Technology*, 12(1):7–20, 2006.

[71] Google Inc. Google App Engine. `http://code.google.com/appengine/`, September 2011.

[72] Google, Inc. Google Apps. `http://www.google.com/apps/`, September 2011.

[73] J. Grethe, C. Baru, A. Gupta, M. James, B. Ludaescher, M. Martone, P. Papadopoulos, S. Peltier, A. Rajasekar, S. Santini, et al. Biomedical informatics research network: building a national collaboratory to hasten the derivation of new understanding and treatment of disease. *Studies in health technology and informatics*, 112:100–110, 2005.

[74] W. Gropp, E. Lusk, and A. Skjellum. Using MPI: portable parallel programming with the message passing interface. 1999.

[75] G. Gruman and E. Knorr. What cloud computing really means. *InfoWorld*, April 2008. Electronic Magazine, available at `http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031`.

[76] R. Guerin, H. Ahmadi, and M. Naghshineh. Equivalent capacity and its application to bandwidth allocation in high-speed networks. *Selected Areas in Communications, IEEE Journal on*, 9(7):968–981, 1991.

[77] F. Guim and J. Corbalan. A job self-scheduling policy for HPC infrastructures. In *Job Scheduling Strategies for Parallel Processing*, pages 51–75. Springer, 2008.

[78] D. Hadas, S. Guenender, and B. Rochwerger. Virtual Network Services For Federated Cloud Computing. Technical Report H-0269, IBM Technical Reports, Nov. 2009.

[79] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. *Grid Computing GRID 2000*, pages 191–202, 2000.

[80] T. Hey and A. Trefethen. The UK e-science core programme and the grid. *Future Generation Computer Systems*, 18(8):1017–1031, 2002.

[81] J. Honeycutt. Microsoft Virtual PC 2004 Technical Overview. *Microsoft, Nov*, 2003.

[82] B. Hong and V. Prasanna. Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 52. IEEE, 2004.

[83] E. Huedo, R. Montero, and I. Llorente. A recursive architecture for hierarchical grid resource management. *Future Generation Computer Systems*, 25(4):401–405, 2009.

[84] D. Jackson, Q. Snell, and M. Clement. Core Algorithms of the Maui Scheduler. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 87–102. Springer Berlin / Heidelberg, 2001.

[85] B. Kandukuri, V. Ramakrishna Paturi, and A. Rakshit. Cloud security issues. In *2009 IEEE International Conference on Services Computing*, pages 517–520. IEEE, 2009.

[86] G. Katsaros, G. Gallizo, R. Kübert, T. Wang, J. O. Fito, and D. Henriksson. A Multi-level Architecture for Collecting and Managing Monitoring Information in Cloud Environments. In *CLOSER 2011: International Conference on Cloud Computing and Services Science (CLOSER)*. Accepted for publication.

[87] G. Katsaros, G. Kousiouris, S. Gogouvitis, D. Kyriazis, and T. Varvarigou. A service oriented monitoring framework for soft real-time applications. In *Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on*, pages 1–4. IEEE.

[88] J. Kay and P. Lauder. A fair Share scheduler. *Commun. ACM*, 31(1):44–55, 1988.

[89] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. Science clouds: Early experiences in cloud computing for scientific applications. *Cloud computing and applications*, 2008, 2008.

[90] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the Grid. *Scientific Programming*, 13(4):265–276, 2005.

[91] K. Keahey and T. Freeman. Contextualization: Providing one-click virtual clusters. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 301–308. IEEE, 2008.

[92] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes. Sky computing. *Internet Computing, IEEE*, 13(5):43 –51, September – October 2009.

[93] A. Kertész and P. Kacsuk. A taxonomy of grid resource brokers. *Distributed and Parallel Systems*, pages 201–210, 2007.

[94] K. H. Kim and R. Buyya. Fair resource sharing in hierarchical virtual organizations for global grids. In *GRID '07: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, pages 50–57, Washington, DC, USA, 2007. IEEE Computer Society.

[95] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the Linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[96] S. D. Kleban and S. H. Clearwater. Fair Share on High Performance Computing Systems: What Does Fair Really Mean? In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 146, Washington, DC, USA, 2003. IEEE Computer Society.

[97] J. Knobloch and L. Robertson. LHC computing Grid technical design report. `http://lcg.web.cern.ch/LCG/tdr/`, September 2011.

[98] D. Kranzlmuller. The future European Grid Infrastructure - Roadmap and challenges. In *Information Technology Interfaces, 2009. ITI'09. Proceedings of the ITI 2009 31st International Conference on*, pages 17–20. IEEE, 2009.

[99] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of Grid resource management systems for distributed computing. *Softw. Pract. Exper.*, 32(2):135–164, 2002.

[100] H. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. Rumble, E. De Lara, M. Brudno, and M. Satyanarayanan. SnowFlock: rapid virtual machine cloning for cloud computing. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 1–12. ACM, 2009.

[101] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent and Grid Systems*, 1(3):169–182, 2005.

[102] A. Lakshman and P. Malik. Cassandra-A Decentralized Structured Storage System. In *Workshop on Large Scale Distributed Systems and Middleware (LADIS)*, 2009.

[103] L. Larsson, D. Henriksson, and E. Elmroth. Scheduling and Monitoring of Internally Structured Services in Cloud Federations. In *Proceedings of IEEE ISCC 2011*, pages 173–178, 2011.

[104] E. Laure, S. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, et al. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006.

[105] E. Laure and B. Jones. Enabling Grids for e-Science: The EGEE Project. *Grid computing: infrastructure, service, and applications*, page 55, 2009.

[106] K. Leal, E. Huedo, and I. Llorente. A decentralized model for scheduling independent tasks in Federated Grids. *Future Generation Computer Systems*, 25(8):840–852, 2009.

[107] B. Leiner, V. Cerf, D. Clark, R. Kahn, L. Kleinrock, D. Lynch, J. Postel, L. Roberts, and S. Wolff. A brief history of the Internet. *Internet Society*, 10, 2003.

[108] LHC Project Webpage. `http://lhc.web.cern.ch/lhc/`, September 2011.

[109] libvirt development team. libvirt: The virtualization api. `http://libvirt.org/`, September 2011.

[110] M. Lindner, F. Galán, C. Chapman, S. Clayman, D. Henriksson, and E. Elmroth. The Cloud Supply Chain: A Framework for Information, Monitoring, Accounting and Billing. In *2nd International ICST Conference on Cloud Computing (CloudComp 2010)*.

[111] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP journal*, 11(1):36–40, 1997.

[112] R. Mach, R. Lepro-Metz, B. Hamilton, S. Jackson, and L. McGinnis. Usage Record Format Recommendation. *Draft Rec-UR-Usage, Global Grid Forum, Usage Record WG, March*, 2005.

[113] Manifesto, O.C. Open Cloud Manifesto. *Availabe online: www.opencloudmanifesto.org/*, 20, 2009.

[114] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation And Experience. *Parallel Computing*, 30:2004, 2003.

[115] P. Mell and T. Grance. The NIST definition of cloud computing. *National Institute of Standards and Technology*, 53(6), 2009.

[116] C. Metz. Interconnecting ISP networks. *Internet Computing, IEEE*, 5(2):74–80, 2001.

[117] Microsoft Corporation. Microsoft Office Live. `http://www.officelive.com`, September 2011.

[118] Microsoft Corporation. Virtual Hard Disk Image Format Specification, September 2011.

[119] Miniwatts Marketing Group. World Internet Usage Statistics News and World Population Stats. `http://www.internetworldstats.com/stats.htm`, September 2011.

[120] C. Morin. Open computing infrastructures for elastic services: contrail approach. In *Proceedings of the 5th international workshop on Virtualization technologies in distributed computing*, pages 1–2. ACM, 2011.

[121] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE transactions on parallel and distributed systems*, 12(6):529–543, 2001.

[122] J. Nakai and R. Van Der Wijngaart. Applicability of markets to global scheduling in grids. *NAS Report*, pages 03–004.

[123] National Science Foundation. US National Science Foundation (NSF). `http://www.nsf.gov/`, September 2011.

[124] Nordic Data Grid Facility. `http://www.ndgf.org/`, September 2011.

[125] NorGrid. `http://www.norgrid.no/`, September 2011.

[126] OnLive, Inc. OnLive.com. `http://www.onlive.com`, September 2011.

[127] Open Grid Forum OCCI-WG. Open Cloud Computing Interface. `http://www.occi-wg.org/`, September 2011.

[128] OpenVZ project team. OpenVZ Wiki. `http://www.openvz.org`, September 2011.

[129] P.-O. Östberg, D. Henriksson, and E. Elmroth. Decentralized, scalable, Grid Fairshare Scheduling (FSGrid). 2011. Submitted.

[130] Parallels. Parallels Optimized Computing. `http://www.parallels.com/eu/`, September 2011.

[131] F. Perez-Sorrosal, M. Patiño-Martinez, R. Jimenez-Peris, and B. Kemme. Elastic si-cache: consistent and scalable caching in multi-tier architectures. *The VLDB Journal*, pages 1–25.

[132] S. Pickles, R. Blake, B. Boghosian, J. Brooke, J. Chin, P. Clarke, P. Coveney, N. González-Segredo, R. Haines, J. Harting, et al. The TeraGyroid experiment. In *Proceedings of the Workshop on Case Studies on Grid Applications at GGF*, volume 10, page 2004, 2004.

[133] R. Piro, A. Guarise, and A. Werbrouck. An Economy-based Accounting Infrastructure for the DataGrid. In *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)*, 2003.

[134] G. Popek and R. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[135] Rackspace, US Inc. Rackspace Cloud. `http://www.rackspace.com/cloud/`, September 2011.

[136] M. Riedel, E. Laure, T. Soddemann, L. Field, J. Navarro, J. Casey, M. Litmaath, J. Baud, B. Koblitz, C. Catlett, et al. Interoperation of world-wide production e-science infrastructures. *Concurrency and Computation: Practice and Experience*, 21(8):961–990, 2009.

[137] D. Ritchie and K. Thompson. The UNIX time-sharing system. *Communications of the ACM*, 17(7):365–375, 1974.

[138] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, S. C. E. Levy, A. Maraschini, P. M. H. Muñoz, G. Toffetti, and M. Villari. RESERVOIR : When one cloud is not enough. *IEEE Computer*, 2011. Accepted.

[139] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galán. The RESERVOIR model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4), 2009. Paper 4.

[140] B. Rochwerger, C. Váquez, D. Breitgand, D. Hadas, M. Villari, P. Massonet, E. Levy, A. Galis, I. Llorente, R. Montero, Y. Wolfsthal, K. Nagin, L. Larsson, and F. Galán. An Architecture for Federated Cloud Computing. *Cloud Computing*, 2010.

[141] L. Rodero-Merino, L. Vaquero, V. Gil, F. Galán, J. Fontán, R. Montero, and I. Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226–1240, 2010.

[142] M. Rosenblum. The reincarnation of virtual machines. *Queue*, 2(5):34–40, 2004.

[143] M. Rothstein. An airline overbooking model. *Transportation Science*, 5(2):180, 1971.

[144] M. Russell, P. Dziubecki, P. Grabowski, M. Krysiński, T. Kuczyński, D. Szjenfeld, D. Tarnawczyk, G. Wolniewicz, and J. Nabrzyski. The vine toolkit: A java framework for developing grid applications. *Parallel Processing and Applied Mathematics*, pages 331–340, 2008.

[145] SAP. SAP Enterprise Resource Planning. `http://www.sap.com/erp`, visited April 2011, September 2011.

[146] J. Schopf. Ten actions when Grid scheduling. In J. Nabrzyski, J. Schopf, and J. Węglarz, editors, *Grid Resource Management State of the art and future trends*, chapter 2. Kluwer Academic Publishers, 2004.

[147] L. Seawright and R. MacKinnon. VM/370-A Study of Multiplicity and Usefulness. *IBM Systems Journal*, 18(1):4–17, 1979.

[148] J. Silvestre. Economies and diseconomies of scale. *The New Palgrave: A Dictionary of Economics*, 2:80–84, 1987.

[149] W. Smith, I. Foster, and V. Taylor. Scheduling with Advance Reservations. In *14th International Parallel and Distributed Processing Symposium*, pages 127–132, 2000.

[150] SNIC. SweGrid - The Swedish GRID Initiative. `http://www.snic.vr.se/projects/swegrid`, September 2011.

[151] B. Sotomayor, K. Keahey, and I. Foster. Combining Batch Execution and Leasing Using Virtual Machines. In *HPDC - The ACM/IEEE International Symposium on High Performance Distributed Computing*, 2008.

[152] B. Sotomayor, R. Montero, I. Llorente, I. Foster, and F. de Informatica. Capacity leasing in cloud systems using the OpenNebula engine. *Cloud Computing and Applications*, 2008, 2008.

[153] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13:14–22, 2009.

[154] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Characterization of backfilling strategies for parallel job scheduling. In *Parallel Processing Workshops, 2002. Proceedings. International Conference on*, pages 514–519. IEEE, 2002.

[155] H. Stockinger. Defining the grid: a snapshot on the current view. *The Journal of Supercomputing*, 42(1):3–17, 2007.

[156] A. Streit, D. Erwin, T. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and P. Wieder. UNICORE - from project results to production grids. In L. Grandinetti, editor, *Grid Computing: The New Frontiers of High Performance Processing, Advances in Parallel Computing 14*, pages 357–376. Elsevier, 2005.

[157] K. Suzaki and D. Walsh. Implementing the Combination of Time Sharing and Space Sharing on AP/Linux. In *Job Scheduling Strategies for Parallel Processing*, pages 83–97. Springer, 1998.

[158] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 111–120. ACM, 2011.

[159] A. Szalay and J. Gray. The world-wide telescope. *Science*, 293(5537):2037, 2001.

[160] M. Szeredi. Filesystem in userspace. `http://fuse.sourceforge.net/`, September 2011.

[161] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency Computat. Pract. Exper.*, 17(2–4):323–356, 2005.

[162] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A grid monitoring architecture. 2002.

[163] J. Tordsson, R. Montero, R. Vozmediano, and I. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. 2010. Submitted for journal publication.

[164] M. Tsugawa and J. Fortes. A virtual network (ViNe) architecture for grid computing. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10–pp. IEEE, 2006.

[165] Ubuntu Community. CloudInit - Community Ubuntu Documentation. `https://help.ubuntu.com/community/CloudInit`, September 2011.

[166] U.S. Department of Commerce. National Institute of Standards and Technology. `http://www.nist.gov`, September 2011.

[167] H. Using Windows. Server 2008 job scheduler. *Microsoft Corporation, Published: June*, 2008.

[168] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 228–235. Ieee, 2010.

[169] L. M. Vaquero, L. Rodero-Merino, J. Cáceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[170] S. Venugopal, R. Buyya, and L. Winton. A Grid service broker for scheduling e-science applications on global data Grids. *Concurrency Computat.: Pract. Exper.*, 18(6):685–699, May 2006.

[171] VMWARE. VMware VMotion: Live migration of virtual machines without service interruption datasheet. `http://www.vmware.com/files/pdf/VMware-VMotion-DS-EN.pdf`, September 2011.

[172] VMware, Inc. VMware vCloud Express. `http://www.vmware.com/solutions/cloud-computing/public-cloud/vcloud-express.html`, September 2011.

[173] A. Waheed, W. Smith, J. George, and J. Yan. An infrastructure for monitoring and management in computational grids. *Languages, Compilers, and Run-Time Systems for Scalable Computers*, pages 619–628, 2000.

[174] J. Walters, V. Chaudhary, M. Cha, S. Guercio Jr, and S. Gallo. A Comparison of Virtualization Technologies for HPC. In *22nd International Conference on Advanced Information Networking and Applications*, pages 861–868. IEEE, 2008.

[175] J. Watson. Virtualbox: bits and bytes masquerading as machines. *Linux Journal*, 2008(166):1, 2008.

[176] A. Weiss. Computing in the clouds. *NetWorker*, 11(4):16–25, 2007.

[177] C. D. Weissman and S. Bobrowski. The design of the force.com multitenant internet application development platform. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 889–896, New York, NY, USA, 2009. ACM.

[178] A. Whitaker, M. Shaw, S. Gribble, et al. Denali: Lightweight virtual machines for distributed and networked applications. Technical report, Citeseer, 2002.

[179] T. White. *Hadoop: The Definitive Guide*. Yahoo Press, 2010.

[180] M. B. Yehuda, O. Biran, D. Breitgand, K. Meth, B. Rochwerger, E. Salant, E. Silvera, S. Tal, Y. Wolfsthal, J. Cáceres, J. Hierro, W. Emmerich, A. Galis, L. Edblom, E. Elmroth, D. Henriksson, F. Hernández, J. Tordsson, A. Hohl, E. Levy, A. Sampaio, B. Scheuermann, M. Wusthoff, J. Latanicki, G. Lopez, J. Marin-Frisonroche, A. Dörr, F. Ferstl, S. Beco, F. Pacini, I. Llorente, R. Montero, E. Huedo, P. Massonet, S. Naqvi, G. Dallons, M. Pezzé, A. Puliato, C. Ragusa, M. Scarpa, and S. Muscella. RESERVOIR - an ICT infrastructure for reliable and effective delivery of services as utilities. Technical report, IBM Haifa Research Laboratory, 2008.

[181] A. Yoo, M. Jette, and M. Grondona. SLURM: Simple Linux Utility for Resource Management. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 44–60. Springer Berlin / Heidelberg, 2003.

[182] J. Yu, S. Venugopal, and R. Buyya. A market-oriented grid directory service for publication and discovery of grid service providers and their services. *The Journal of Supercomputing*, 36(1):17–31, 2006.

[183] S. Zanikolas and R. Sakellariou. A taxonomy of grid monitoring systems. *Future Generation Computer Systems*, 21(1):163–188, 2005.

I

Paper I

Distributed Usage Logging for Federated Grids*

Erik Elmroth and Daniel Henriksson

Dept. Computing Science and HPC2N, Umeå University
SE-901 87 Umeå, Sweden
{elmroth, danielh}@cs.umu.se
http://www.cs.umu.se/ds

Abstract: We present a non-intrusive solution to the increasingly important problem of shared logging for overlapping and federated Grid environments. The solution addresses three usage scenarios of hierarchical Grids, mutual cross-Grid resource utilization, and federated Cloud computing infrastructures. The approach is evaluated by extending the existing SweGrid Accounting System (SGAS) with a light-weight component that makes the system applicable to a wide range of usage scenarios. The proposed architecture is characterized by its simplicity, flexibility, and generality, and the new key component by its non-intrusiveness, flexibility, and ability to manage high load. We present requirements derived from three usage scenarios, and also include an in-depth description of the architecture and design, as well as the implementation and performance evaluation of a new component written for use with SGAS. We conclude from a performance evaluation that the sharing of usage data is not likely to be a limiting performance factor even in large-scale Grid scenarios.

Key words: accounting, shared logging, grid computing, SGAS, federated grids, federated clouds

Future Generation Computer Systems 26 (2010) 1215–1225

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Distributed usage logging for federated Grids

Erik Elmroth, Daniel Henriksson *

Department of Computing Science & HPC2N, Umeå University, SE-901 87 Umeå, Sweden

ARTICLE INFO

Article history:
Received 20 May 2009
Received in revised form
14 December 2009
Accepted 1 February 2010
Available online 6 February 2010

Keywords:
Accounting
Shared logging
Grid computing
SGAS
Federated Grids
Federated Clouds

ABSTRACT

We present a non-intrusive solution to the increasingly important problem of shared logging for overlapping and federated Grid environments. The solution addresses three usage scenarios of hierarchical Grids, mutual cross-Grid resource utilization, and federated Cloud computing infrastructures. The approach is evaluated by extending the existing SweGrid Accounting System (SGAS) with a light-weight component that makes the system applicable to a wide range of usage scenarios. The proposed architecture is characterized by its simplicity, flexibility, and generality, and the new key component by its non-intrusiveness, flexibility, and ability to manage high load. We present requirements derived from three usage scenarios, and also include an in-depth description of the architecture and design, as well as the implementation and performance evaluation of a new component written for use with SGAS. We conclude from a performance evaluation that the sharing of usage data is not likely to be a limiting performance factor even in large-scale Grid scenarios.

1. Introduction

Shared logging is an increasingly important problem for accounting and usage tracking in overlapping and federated Grid environments. Large scientific projects such as the Large Hadron Collider [1] have outgrown the capacity of any existing single Grid, making federations of Grids more and more common. We present a flexible architecture for usage logging in federated Grids, and also describe the design of a self-contained light-weight mediator component that abstracts the sharing of usage data from the systems involved by operating only on methods available to regular clients. This approach is evaluated by implementing a component primarily targeting the SweGrid Accounting System (SGAS) [2,3]. Background information on SGAS is provided in Section 2 to set the context in which this new component operates.

The accounting architecture is designed to be non-intrusive to existing accounting system installations and to meet the requirements of the following three usage scenarios:

- *Hierarchies of (parts of) Grids.* A common usage scenario in international research collaborations is that a large international organization or project is allocated fractions of several (national) Grids. In addition to the requirements for logging for internal reporting in each Grid, the international organizations need to log their combined usage on all Grids in the federation.
- *Mutual cross-Grid resource utilization.* In order to efficiently meet load peaks in (e.g., national) Grids, the ability to perform inter-Grid resource exchange is of increasing interest. This in turn gives rise to requirements on inter-Grid logging of resource usage, as both the producing and consuming organizations need to document all cross-Grid resource consumption.
- *Cost compensation in federated Cloud computing infrastructures.* In federated Cloud infrastructures, such as RESERVOIR [4], it is crucial for the different sites forming the infrastructure to accurately exchange usage data.

By analyzing the proposed architecture in light of these usage scenarios, we show that the solution is flexible and adaptable enough to join arbitrary structures of hierarchies and collaborations. An example of a hierarchy even more complex than the above usage scenarios is illustrated and discussed in Section 6.

The rest of this paper is organized as follows: Section 2 presents the background, including an overview of the SGAS accounting system. Section 3 elaborates on the three major usage scenarios motivating this research. The requirements of the new support for logging in a federation of Grids is presented in Section 4. The overall architecture is presented and motivated in Section 5. The design and implementation of a component realizing the architecture are presented in Section 6 followed by a performance evaluation in Section 7. Future work and some concluding remarks are given in Sections 8 and 9, respectively.

2. Background

2.1. SGAS

The SweGrid Accounting System (SGAS) [5,2,3] is a capacity allocation management system for Grid environments. Typically, SGAS is used to maintain a credit-based allocation model where

* Corresponding author. Tel.: +46 705709331; fax: +46 907866126.
E-mail addresses: elmroth@cs.umu.se (E. Elmroth), danielh@cs.umu.se (D. Henriksson).

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.02.001

projects are granted allowances to be spent across resources on the Grid. These allowances are collectively enforced by the resources in a soft, real-time manner.

SGAS is a flexible service-oriented accounting system, designed for use in a wide range of Grid environments. Independent evaluations have found SGAS superior to existing alternative accounting systems, in particular with respect to interoperability, ability for integration, portability, accounting beyond one community, standards support, security, fault tolerance, accuracy, administration, and verification [6,7]. The system is implemented using the Globus Toolkit 4 (GT4) [8].

SGAS is designed for flexibility and adaptability to be used in widely different usage scenarios. To achieve this, the system consists of three self-sustaining components, each responsible for a distinct and independent part of the accounting procedure [2]:

- *Bank*: The bank service manages usage quotas and aggregated resource consumption for a set of resources facilitating coordinated quota enforcement on the Grid. Credits are pre-allocated to users and partly consumed as users run Grid jobs. An abstract currency referred to as *Grid Credits* is used within the system, making it possible to change the policies for credit allocation and if applicable change the mapping from Grid Credits to real currencies, without affecting the accounting system. However, the Bank component is only concerned with handling the accounts and the abstract Grid Credits, and is unconcerned with the type of resources being used or how utilization of various types of resources is mapped to Grid Credits.
- *Logging and Usage Tracking Service (LUTS)*: XML-based usage records for completed jobs are published by the Grid sites and stored in the LUTS through a Web service interface. The LUTS uses an XML database back-end to store the usage records in a native format.
- *Job Account Reservation Manager (JARM)*: The JARM component acts as a bridge between local resources and the Grid-wide accounting context. This interaction is done by intercepting calls between the Grid middleware component dealing with job submission and the local resource management systems. When such calls are intercepted by the JARM, an amount of credits sufficient to cover the expected resource consumption of the job are reserved prior to the job execution. The JARM is also responsible for mapping usage to Grid Credits using configurable policy managers. Local resource policies specify if jobs without a successful reservation should be allowed to execute or not (or possibly execute with a lower priority), depending on, for example, the current load on the system. SGAS supports different JARM implementations, making it possible to adapt the accounting system to Grid infrastructures with different job submission mechanisms.

A typical usage scenario for an SGAS-equipped system starts with a job arriving at a Grid job manager, for example a WS-GRAM or ARC GridManager, on a resource. Before the job is forwarded to the local resource manager for execution, the JARM makes a reservation of credits in the Bank. The amount of credits reserved is typically based on a user-specified estimate of the maximum amount of resources required to complete the job. If there are sufficient credits in the Bank, the job is executed. Once the job has completed, sufficient credits to cover the definite amount of consumed resources is deducted from the previous reservation, and the surplus of reserved credits (if any) is released to the Bank account for later use. At this stage, a usage record containing the details of the job execution is stored within the logging component (LUTS).

A Grid system can use one or both of the Bank and LUTS components. A typical example where the LUTS is used independently of a Bank is usage scenarios where only usage logging is required, for instance when using a periodic billing approach based on the accumulated usage information stored in the LUTS. In this scenario real-time quota enforcement is not required, and therefore the system can run without the Bank component.

2.2. Usage records

Resource sharing between different Grid sites, portability, and inter-Grid interoperability can be considerably facilitated by using a common format for basic accounting and usage data, such as the usage records format (OGF-UR) [9] proposed by the Open Grid Forum (OGF) [10]. The proposal covers the basic building blocks and representation of accounting records.

The SGAS LUTS is capable of logging any kind of XML data supplied and is not dependent on the data conforming to the OGF-UR format. The implementation of the component described in Section 6 makes use of one of the properties specified in the OGF-UR recommendation called *RecordIdentity* to filter out duplicate usage records. By using a time stamp attribute added to each record upon insertion into the LUTS, it is also possible to identify recently added records.

As discussed in Section 3.3, the OGF-UR might not be applicable for all type of "jobs", but as long as the usage data is represented in XML and contains the required RecordIdentity property, any kind of present and future representation of usage records is compatible with this logging and federation approach.

3. Usage scenarios

Three different usage scenarios have been identified to provide a framework for this research. The use cases are presented without relating to the architecture proposed in Section 5, and are general to the concept of shared logging within federated Grid and Cloud environments.

Each use case is presented in the following subsections, covering both the general case and a specific scenario for each of the cases. Section 6.2 reviews and evaluates the design of the proposed federation architecture in the context of these use cases.

3.1. Hierarchies of Grids

A common usage scenario in international research collaborations is that of a large international organization or project which is allocated fractions of several (national) Grids. Each of these Grids needs to perform usage logging for their own reporting and the international organization needs to log their usage on all Grids. In this scenario, it is crucial that the local resource owners maintain control of the local resources to preserve the independence of each collaborating Grid, and the ability to maintain functional accounting of local resource usage within the local Grid. A special case of this scenario is when, e.g., a local site manages its own logging for some or all of its resources, and subsequently wishes to share portions of the usage records with other sites participating in the Grid.

An example of such collaboration is the cooperation between the Nordic DataGrid Facility (NDGF) [11] and SweGrid [12]. NDGF is a consortium where SweGrid is one of the participants, and a subset of the jobs run on SweGrid are run by NDGF users. In this scenario, the usage data from SweGrid jobs associated with NDGF for the accounting purposes of both SweGrid and NDGF. Moreover, SweGrid usage records concerning jobs unrelated to NDGF must be kept private and unexposed. Since SGAS is already used in both SweGrid and NDGF, changes to the existing SGAS environments must be kept to an absolute minimum when adding the shared logging capabilities.

3.2. Mutual cross-Grid resource utilization

In order to efficiently meet load peaks in Grids, the ability to perform inter-Grid resource exchange is of increasing interest. This in turn gives rise to requirements on cross-Grid logging of resource usage, as both the producing and the consuming organization need to document all cross-Grid resource consumption. Unlike the previous usage scenario, Grids in this scenario are not hierarchically structured and so the sharing must be done between peers in a mutual fashion. From the point of view of each Grid, it is important to monitor all local jobs run on the local infrastructure, all local jobs run on any remote infrastructure, and also all remote jobs run on the local infrastructure.

Consider, for instance, the two national Grids of Sweden and Norway, SweGrid and NorGrid [13], forming an inter-Grid resource exchange agreement. The requirements are that the usage records related to this capacity exchange should be available for both Grids, while it is important to maintain confidentiality and protect usage records that are not associated with cross-Grid usage.

3.3. Cost compensation in federated Cloud computing infrastructures

Cloud computing [14] is a term used to describe an infrastructure providing computational resources on demand, with built-in flexibility (typically referred to as *elasticity*) to react to changes in utilization and provide scalability to meet the new load. Many of the technologies and components developed for Grid computing can be reused for Cloud computing.

Future Cloud infrastructures are likely to be composed of more than one collaborating site, creating a federated infrastructure that poses new challenges for accounting of resource usage [15]. Services run on Cloud infrastructures are often different from Grid jobs as they do not have a limited execution time and their resource consumption per time unit may vary. This makes them unsuitable for post-completion accounting and could also make the usage records hard to represent using the OGF-UR format, since these usage records are job-centric. Services, or distinct components as a part of a service, can usually run on remote sites in different domains, or be migrated between sites during the course of their execution. The migration of a running service component can potentially fragment the usage data across multiple sites. Components of Cloud services are normally run inside virtual machines to make the migration possible despite heterogeneous environments. This means that accounting has to be done for each virtual machine, or even for the combined consumption of several virtual machines running on different administrative domains. This can infer variations in the structure and shape of the accounting data, increasing the importance of flexibility with regards to the format and content of each record.

One example of a federated Cloud infrastructure is RESERVOIR [4]. In RESERVOIR, sites form framework agreements allowing the sites to deploy service components on (or migrate services to) another site if the amount of available local resources are insufficient. The site from which the service originates is denoted the *primary* site, and any collaborating site running one or more service component is called a *remote* site. At any time during the service execution it is crucial for the primary RESERVOIR site to accurately retrieve relevant usage data from collaborating remote RESERVOIR sites. Using RESERVOIR, usage information for each component in a service needs to be periodically recorded by the site currently running the component, and this information has to be aggregated at the primary site to create unified usage data for the entire service.

4. Requirements

The use cases in Section 3 form a base of requirements for the resulting architecture. The requirements derived from the use cases are complemented with requirements affecting performance and security, and the following list items summarizes the requirements on the federated logging solution:

- Req. 1 CARDINALITY: The architecture must enhance the normal functionality of local logging by usage reporting with different cardinality between information producers (Grids) and logging databases (one-to-one, one-to-many, many-to-one, and many-to-many).
- Req. 2 NON-INTRUSIVE: The shared logging functionality should be an optional and non-intrusive addition to already running installations of accounting systems. Changes to existing system environments should be kept to an absolute minimum.
- Req. 3 BATCH-WISE PROCESSING: The system should allow for batch-wise processing of usage records in order to minimize overhead and increase throughput.
- Req. 4 DUPLICATION: For consistency and performance reasons, the risk of duplication of usage records in the same database should be minimized.
- Req. 5 ISOLATION: Use of shared logging should not pose a threat to the exposure of records not associated with the resource exchange, and only the relevant records must be made available to collaborating parties.
- Req. 6 DATA VARIATION: The system must be flexible enough to handle variations in information stored for accounting.
- Req. 7 AUTONOMY: The system must allow resource owners and individual Grids to preserve autonomy.
- Req. 8 MULTIPLE FEDERATIONS: A Grid must be able to participate in more than one federation at a time. For example, a single Grid should be able to participate in two or more hierarchical Grids and a mutual exchange setup at the same time.

5. Architecture

Four different architectural approaches have been evaluated with respect to the requirements presented. The approaches are briefly described below, together with some details about why the selected approach has been found advantageous over the other three. In the description below, the term *database* is widened to include also entire logging components such as the SGAS LUTS.

- *Common centralized database.* In this approach, each collaborating Grid defers its own usage logging database and instead performs all usage logging in a common database used by all participating Grids. In effect, each resource directly sends all logging data to a database common for the collaborating Grids instead of to a database specific to each Grid. This approach clearly facilitates the aggregation of usage data independently of which Grid has been used. However, it violates several of the above requirements, including that it affects the individual Grids' autonomy, it poses risks to the privacy of data not relevant to the federation, and it has serious scalability limitations as different Grids enter multiple different collaborations or takes part in several concurrent exchanges or hierarchical arrangements.
- *Resource-driven shared logging.* Logging into multiple databases can be done by having each resource directly perform usage logging to more than one target database. The databases can be individually selected for each resource consumption, and the additional logic to achieve this can be placed at the resources. Although this approach can be developed to meet most of the requirements, it has a major weakness in the complexity added to each resource. This in turn has consequences in the form of additional administrative burdens for each individual Grid resource when new Grid collaborations are formed, as well as performance penalties at the time of job submission. Another

drawback of this approach is that it requires modifications of the components within the accounting system that performs the usage logging to support logging to a changing set of databases. However, it should be noted that if Grid-wide usage quota enforcement also is to be performed towards multiple instances, this approach may have to be reconsidered.

- *Client-level shared logging.* With client-level shared logging we refer to relying on a single client to perform reading from a set of source databases and posting the results to one or more target databases. The client can reuse existing client interfaces to read and write the data, and this way neither the source nor target databases will be aware of the sharing taking place. This means that the source and target databases can remain unaware of the sharing procedure, and no alternation of the databases is required. Another advantage is that several kinds of databases can be included in the sharing procedure if they offer the same kind of interfaces, or if the component responsible for the interactions is adapted to support the different technologies. The main drawback with this approach is that the data to be shared between a set of databases is sent twice, once from the source database to the client and once from the client to the target database. This is quite similar to, e.g., how RSS [16] feeds can be combined to form a single digest (see, e.g., [17]) where only those matching a specified pattern are included. The digested RSS feeds and the client-level shared logging approach share the same weakness since the data is being sent (at least) twice, but they also share the same advantages since data can be combined and filtered from several different sources while still keeping the final consumer (or target database) unaware of this procedure.
- *Materialized database views.* Another approach with similarities to the client-level shared logging described above is that of materialized views [18], which is an established technique to create database views that can be updated when the underlying data changes. This can also be applied to distributed databases, making it possible to host a specific materialized view in a single database which would have similar properties to that of a target database in the approach above, since data from several sources matching a specified pattern in both cases would be sent to a unified location even before it is requested. The main drawback with these approaches is that the materialized views approach is less generic as it requires an underlying database software with support for generating and maintaining these kind of views. It is also considerably harder to make several different accounting systems use the same database software as opposed to creating a client-level shared logging component capable of communicating with several different interfaces.

Based on the above analysis, the client-level shared logging approach is determined to be the most suitable approach in order to achieve the distributed usage logging for federated Grids. Compared to the other alternatives, this approach has substantial advantages in terms of decentralization, non-intrusiveness, Grid autonomy, and ease of integration with already running accounting system installations. The performance, efficiency, and stability of the different solutions are of less importance for this decision, as we argue that the above non-technical factors are more relevant in order to meet the requirements. However, we still have to ensure that performance is not a bottleneck, and for that reason a performance evaluation of the suggested approach can be found in Section 7.

Of the non-technical factors listed above, the most central one is the non-intrusiveness and abstraction offered by doing the interaction with the databases using the normal client interfaces for reading and writing records. This offers great advantages in flexibility as the underlying specifics of each database component can be ignored, and also makes it possible to support several different accounting system technologies by implementing technology-specific clients (or using a common interface, if possible). Widespread use of a common format (such as the OGF-UR recommendation) makes it likely that the usage records are represented in the same format across different accounting systems. If this is not the case for a particular accounting system, the technology-specific parts of the client-level shared logging component can be extended to perform a transformation of the data into the common format before transferring the data to the target database. The drawback of having to send the data twice can be mitigated by co-hosting the client with either the source database or the target database. These alternatives are elaborated on later in upcoming sections.

In our case, we have chosen SGAS as the accounting system in which to implement the client-level approach. The remaining sections describes and evaluates a proposal for an architecture that is implemented using SGAS. The proposed architecture is presented in the next subsection, and its ability to meet the requirements from Section 4 is further investigated in Section 6.2.

5.1. Architecture for Client-level shared logging in SGAS

In SGAS, usage records are normally stored within a LUTS component running within the administrative domain of the Grid. The proposed architecture extends the existing SGAS logging functionality with an additional component that can filter out and forward usage record between sets of LUTSes at regular intervals. We refer to this component as the LUTS federation component (LUTSfed). This component is designed to act as a client from the point of view of each LUTS, and reuse the client mechanisms for reading and writing records.

Two of the key features of the proposed architecture are simplicity and flexibility. The architecture is designed to be easy to comprehend, modify, and extend in the future. By reusing existing functionality we can rely on the native database of the source LUTS to compose result sets of usage records to be sent along to the target LUTSes, the components in the architecture of the LUTSfed are kept minimalistic and light-weight. This has a positive impact on scalability and performance, and all future improvements to the reused code in SGAS will be automatically adopted by the federation component.

By using specific queries when reading from the source LUTS, the set of usage records to be shared can be limited to a subset of the available records, e.g., those related to a specific project or all records that have a specific tag for this purpose. The database backend of the LUTS natively supports XPath [19] queries. XPath is able to select sets of XML instances based on the value of any attribute or element in the instance, which means that the queries can be constructed to target any part of the usage records, regardless if these records comply with a common format or not.

Since the LUTSfed component is independent of any specific LUTS instance, it can (given sufficient permissions) operate on a set of source LUTSes and send the resulting data along to a set of target LUTSes. This way, the component supports different cardinality between the sources and targets. Fig. 1 shows an overview of the interaction between the new LUTSfed component and the LUTSes of two different existing SGAS installations. Of course, the LUTSfed component can also be used to share usage records within a Grid, e.g., from a LUTS running at a local site to another LUTS responsible for Grid-wide accounting.

The LUTSfed can be hosted and maintained either within the administrative domain of the source LUTS, the target LUTS, or by a third party. Security is maintained regardless of the environment in which the LUTSfed component is run by using the existing security architecture from GT4 including the fine-grained authorization support contributed by SGAS [3]. However, accessing usage records published by other users requires super-user privileges, and since sharing such privileges could possibly violate the isolation of usage

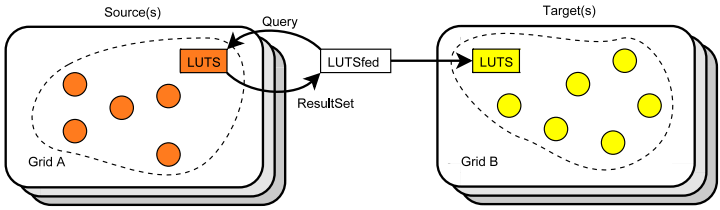


Fig. 1. An overview of the LUTSfed interaction with existing LUTSes. The federation component reads usage records from one or more sources, and publishes the results to one or more targets.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entries>
    <entry source="https://130.239.40.45:8443/wsrf/services/sgas/LUTS">
        <xpath_query>SGASProjectName='alice'</xpath_query>
        <target>https://130.239.40.45:8442/wsrf/services/sgas/LUTS</target>
    </entry>
</entries>
```

Fig. 2. A sample LUTSfed entry file. Each entry contains a source, a query, and one or more targets.

records not being relevant to the federation, it is recommended to run the LUTSfed component in conjunction with the source LUTS.

6. Design and implementation

As stated in Section 4, one requirement for the LUTSfed component is to harmonize with other components of the accounting systems. In the SGAS case, this is achieved by preserving the service-oriented approach used in the SGAS design, and by implementing the LUTSfed component as a stand-alone Web service capable of running either in its own Web service container or within the same service container as other SGAS components.

When the LUTSfed is initialized it reads user configured *entries* from the *entry file*. A simple example of an entry file is illustrated in Fig. 2. Each entry specifies a source LUTS, an XPath query, and one or more target LUTSes. This enables shared logging using multiple queries with multiple sources and targets using a single LUTSfed component.

SGAS provides functionality for periodic task scheduling, very much like a server-side *cron* (scheduled periodical execution) mechanism. On every iteration, each entry from the entry file is processed as shown in Fig. 3, in the order they are specified. The length of the period between executions can be adjusted in a configuration file, and this file is automatically reloaded without restarting the LUTSfed. The LUTSfed also reuses existing SGAS code for querying and publishing records. Reuse of these methods ensures that the interaction with the LUTS components is the same regardless of whether records are published using a JARM, a client program, or as a part of the sharing procedure.

The LUTS automatically adds a *creationTime* attribute to records when they are added to the database. To avoid conflicts, this data has to be removed from the usage records before they are forwarded to a target LUTS. This removal of the creation time attribute is required since a LUTS does not accept records where this attribute already exists.

The querying mechanism in LUTSfed automatically augments the queries with time constraints. This can reduce the size of the result sets considerably as only records added since the last interaction are selected. This is achieved by appending an additional XPath constraint, targeting the LUTS creationTime attribute, to the query specified by the user. The time stamp used in this filtering is the time stamp of the most recent record previously processed for this entry and target, and LUTSfed stores these time stamps persistently by using a time stamp data access object (DAO).

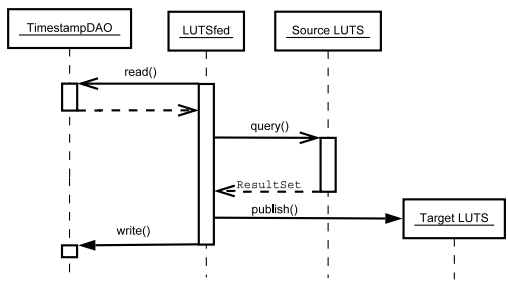


Fig. 3. Overview of the main LUTSfed interactions when evaluating an entry. The LUTSfed component sends a query to the source LUTS and forwards the result set to the target LUTS. The query includes the time stamp of the most recently processed record to avoid processing older records several times, and these time stamps are managed by the time stamp DAO.

Result sets obtained from the source LUTSes are processed batch-wise to reduce the memory footprints related to creating and parsing potentially large amounts of SOAP wrapped data. A result of the above mentioned time stamp filtering approach is that at most one redundant batch of records is processed after a crash.

The LUTSfed component receives the records as plain text representing XML. This data does not have to be unmarshalled (and subsequently marshalled) in the LUTSfed, which considerably reduces the load on this component.

6.1. SGAS improvements related to its use in federations of Grids

A few general improvements to SGAS have been made during the development of the architecture of the LUTSfed component. These improvements are valuable also in environments not using the LUTSfed component, but are of particular interest in this context as they eliminate the risk of duplication of usage records in the LUTS. In Section 7, it is shown that the performance impact of these changes is negligible.

- Duplicated records can, e.g., due to misconfiguration or restart of a client, be published in a target LUTS. To compensate for this, the LUTS database has been modified to allow only one record with the same RecordIdentity attribute (see Section 2.2). XML instances that do not contain this attribute are not affected. Moreover, in case alternative ways of publishing entries to the

LUTS become available in the future, the logging system can with this improvement avoid duplicates without relying on the client(s) to guarantee uniqueness of input data.
- Another modification related to the one mentioned above is that the LUTS database now uses a unique time stamp per record, instead of a unique time stamp per batch of records. This is achieved by using simple mutual exclusion directives when assigning the time stamps, ensuring that the time stamp is unique. In additions to the effects already mentioned, this also makes the architecture more resilient to crashes during large batch processing.

6.2. Applicability to use cases

This section reviews the design and implementation of the proposed architecture for shared logging in general and the LUTSfed component in particular, in terms of the use cases specified in Section 3. The requirements specified in Section 4 are fulfilled as follows:

- Req. 1 CARDINALITY: The logging procedure is enhanced with inter-site logging with support for different cardinality (one-to-one, one-to-many, many-to-one, and many-to-many) defined in the entry file (shown in Fig. 2). For environments with strict security requirements, many-to-one and many-to-many can be achieved by using several LUTSfed instances.
- Req. 2 NON-INTRUSIVE: The LUTSfed is an optional and non-intrusive addition to already running instances of SGAS, as the component can be added without affecting already existing components. It can be run in the same service container as other components or in a dedicated one. No modifications to the existing SGAS environment are required.
- Req. 3 BATCH-WISE PROCESSING: In LUTSfed, usage records are processed batch-wise, reusing existing SGAS code for this purpose.
- Req. 4 DUPLICATION: The LUTSfed implementation presented avoids redundant record processing by augmenting the queries sent to the source LUTS with time stamps of the most recently processed record for each entry and target.
- Req. 5 ISOLATION: The LUTSfed component reuses the security framework employed by the other SGAS components. By running the LUTSfed component in the same administrative domain as the source LUTS, the sharing can be easily configured to ensure that only relevant records are made available to collaborating parties.
- Req. 6 DATA VARIATION: Any XML-based usage records that contains a RecordIdentity element can be managed by the LUTSfed component without any limitations on functionality. The queries can be designed to filter on any available element or attribute of the relevant usage records.
- Req 7 AUTONOMY: The LUTSfed only makes use of Grid-specific LUTSes using their native interfaces and without any requirements that affects the Grids' or resource owners' autonomy.
- Req 8 MULTIPLE FEDERATIONS: The inherent flexibility of the LUTSfed component makes possible to form arbitrary structures and hierarchies of LUTSes, while still hiding this sharing procedure from the underlying accounting system.

To meet the requirements of the three usage scenarios, the presented architecture can be configured as follows.

In the hierarchical Grid usage scenario, shared logging can be used to easily aggregate the usage records of all jobs related to the multi-Grid project run on any participating site, while the participating site can share the appropriate usage records without exposing data about unrelated jobs. This can be achieved by using a simple hierarchical structure where the participants relay usage records to a centralized LUTS using their own instance of the LUTSfed service (see Fig. 4).

However, if a sufficient trust relation is established between the sites, or if the requirement of total privacy for usage records not relevant to the federation can be relaxed, there is an alternative configuration for this use case. By hosting the LUTSfed component at a third (trusted) party or within the site of the target LUTS, the same LUTSfed component can be used for extracting data from all contributing Grids. This configuration is illustrated in Fig. 5.

The inter-Grid resource utilization use case can be solved using the same setup as initially discussed in the hierarchical Grid usage scenario, but the drawback is that usage records related to the inter-Grid utilization would be published in a separate LUTS compared to the usage records of unrelated jobs. This could complicate report generation, since records from more than one LUTS have to be collected and processed. Another approach is to cross-link the existing LUTSes of the different Grids, and share usage records between them. This approach is illustrated in Fig. 6. Since participants run their own LUTSfed components, records that are not related to the federation are ensured to remain private and unavailable from other sites.

The scenario concerning cost compensation in federated Cloud computing infrastructures (Scenario 3) can be solved using the same setup as shown in Fig. 4, but has higher demand on flexibility because services can be migrated between sites during their execution. The LUTSfed can be configured to filter usage records on any attribute or value identifying the site of origin, and any matching usage records are then transferred back to and aggregated in the LUTS of the originating site.

Notably, if sufficient trust relations are established, e.g., via a common trusted party, the LUTSfed-based architecture allows for great flexibility. For example, in a scenario where SweGrid and NorGrid both performs cross-Grid resource utilization and (each of them individually) contributes resources to NDGF, an architecture can be set up with anything from four LUTSfed instances (each having one of the roles illustrated above) to one LUTSfed taking care of all sharing of usage data.

A more complicated usage scenario, along with a proposed solution, is shown in Figs. 7 and 8. The specifications for this scenario are as follows:

- Grid A and B have a mutual exchange of usage records across Grid boundaries, and also participate (possibly with several other Grids) in Collaboration 1.
- Collaboration 1 may in turn be a part of other larger organizations, so usage records originating from jobs run on behalf of Collaboration 1 must be aggregated on and managed collectively.
- Grid C also is a part of the scenario, but does not have any formal collaborations with A or B.
- Each Grid have several sites contributing resources (but only those for Grid A, Site X–Z, are covered in more detail).
- Site X has no local need for storing usage info, and can share any usage records on a Grid-wide level.
- Site Y and Z contribute only with part of their resources to Grid A, and Site Z also contribute with some resources to Grid C.

The above usage scenario can be managed by using several instances of LUTSes and LUTSfed components. Fig. 7 shows the proposed solution at the inter-Grid level, and here each Grid and the collaboration hosts a LUTS of their own to manage records common for organization. Grid A and B also host LUTSfed components that manage both the mutual sharing of records with each other, and also the sharing of usage records related to Collaboration 1. The collaboration also hosts a LUTSfed component to manage the sharing of records with any larger projects or organizations the collaboration is a part of.

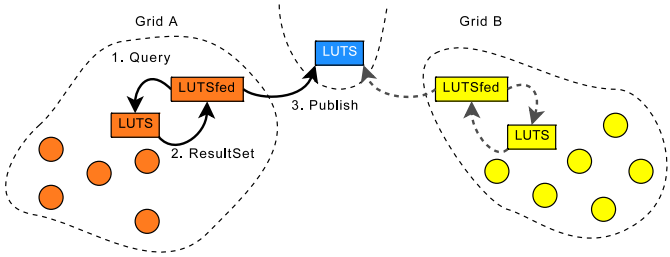


Fig. 4. Overview of a shared logging scenario in the context of the Grid hierarchy usage scenario (Scenario no. 1). Each Grid that is to forward accounting data to the higher-level LUTS runs their own LUTSfed component, forwarding (a subset of) their usage records to a centralized LUTS.

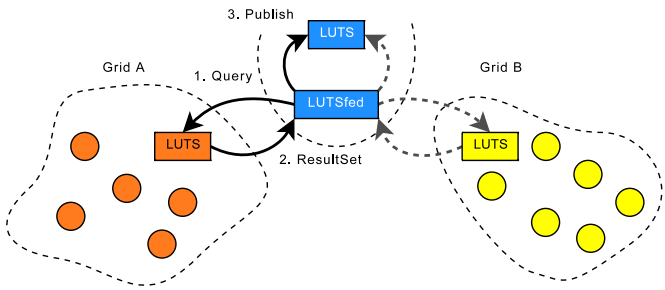


Fig. 5. An overview of a second approach to the Grid hierarchy usage scenario (Scenario no. 1). A LUTSfed component within a trusted environment queries the LUTSes of participating Grids.

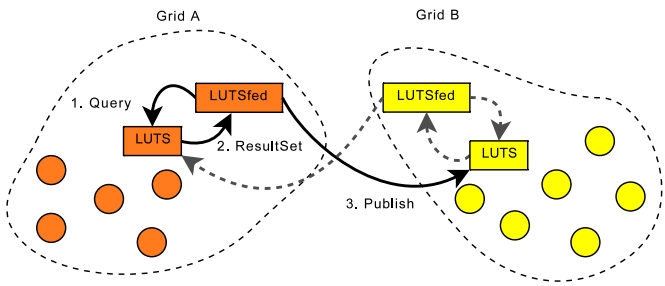


Fig. 6. In the cross-Grid utilization usage scenario (Scenario no. 2), each participant run its own LUTSfed component responsible for transferring usage records to the LUTS running on any cooperating Grid.

The internal setup of Grid A is shown in Fig. 8. Since Site X has no need for local logging, the usage records from its resources are sent directly to the LUTS of Grid A. Site Y has a LUTS of its own to manage the local logging, but allows the LUTSfed component of Grid A to read and share the relevant records. Site Z also contributes to Grid C, and therefore a local LUTSfed component is responsible of sharing records with the LUTSes of both A and C depending on which Grid is responsible for the job.

7. Performance evaluation

The proposed architecture for shared logging provides great flexibility for configuration in terms of the number and placement of LUTS and LUTSfed components. Hence, the performance of the system follows directly from the performance of the individual components. The performance of the LUTS component is thoroughly analyzed in [2]. In this section, we present a performance analysis for the LUTSfed component and discuss the overall system performance.

The performance evaluation of the LUTSfed component has been measured in a test environment with 17 identical machines, each equipped with an Intel Core 2 Quad Q9300, 4 × 2500 MHz CPU, 4 GB of memory, and connected to a shared 100 Mbit network with several switches (some of which are connected to a Gigabit backbone, thus allowing the total amount of data concurrently being transferred to exceed 100 Mbit). Each node is running Debian 5 (kernel 2.6.27.3) as the operating system, and GT4 4.0.8 as the service container.

All computers in the test environment run components based on SGAS version 2.2.0. Eight nodes were set up to host the source LUTSes, and another eight were configured to host the set of target LUTSes. The LUTSfed component was installed on the 17th machine, and the total execution time of each test was measured from the time the query was initiated in the LUTSfed component to when the final insert was completed in all target LUTSes.

The LUTSfed component was configured to transfer all records matching a simple query, using an entry file similar to the one

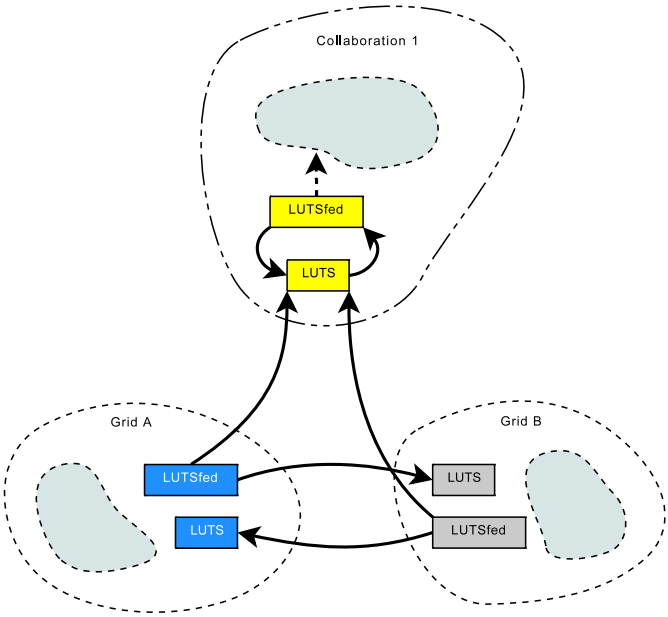


Fig. 7. Grids A and B share usage records with each other, and also with a joint Collaboration 1. The Collaboration can in turn forward the records elsewhere. Note that Grid C is omitted in this illustration.

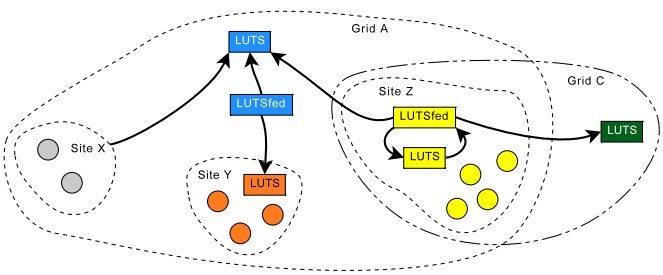


Fig. 8. Sites X, Y, and Z all contribute to Grid A, and Site Z also contributes to Grid C. Site X publishes records straight to the LUTS of A. Site Y hosts a LUTS of its own, so relevant records are transferred to the LUTS of A using the common LUTSfed instance. Similarly, Site Z shares relevant records with both A and C, but in this case using a local LUTSfed instance.

shown in Fig. 2, which was modified to incorporate up to eight sources and targets, respectively. Before each test, the sources were loaded with pre-generated sets of usage records, where different subsets of the records matched the specified query. Each usage record was based on a single authentic usage record from an existing SGAS environment, with a modified identity to make each record unique.

In the evaluation, we present measurements covering both many-to-one scenarios where records from several source LUTSes are sent to a single target LUTS, and one-to-many where one source LUTS shares its records with several target LUTSes. Pure write and read operations in the LUTSes were also measured to give an indication of the maximal performance of these operations. In these measurements, reading 100,000 records all matching a specified query took 218 s, and writing 100,000 unique records took 259 s. (Also see Fig. 11 for comparisons between times for writing and record sharing.) Note that when using the LUTSfed component, the component first reads a small amount of records from the source, and then publishes these records to the target using a separate thread. Therefore, reading from the source and writing to the target can be performed in parallel.

The result of the first set of tests is summarized in Fig. 9. Here, records from a single source LUTS are transferred to one, two, four, or eight different target LUTSes using the same query. Tests where performed using sets of usage records with 1%, 10%, and 100% shares of records matching the XPath query, respectively. Fig. 9 shows that the time to transfer data from a single source to several targets is at most linear with respect to the number of targets. The fact that the time is less than proportional to the number of targets is explained by the source's ability to cache data to be transferred, which here is possible as the same query used to filter out data to be transferred to all sources. Writing operations in the target LUTSes can also be done in parallel to read operations in the sources, which increases the general throughput of the system. For very large amounts of data, the limited network capacity will become the limiting factor.

Fig. 10 shows the results of transferring (sharing) records from many different sources to a single target LUTS. Note that the same

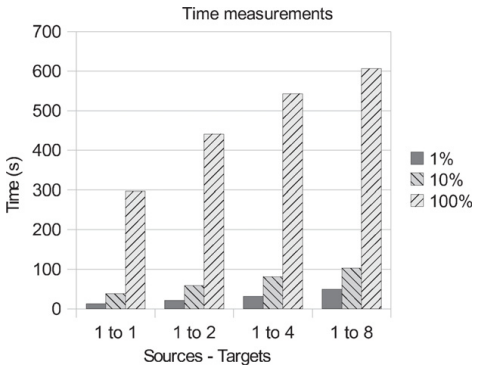


Fig. 9. Usage records are shared between a single source LUTS and several target LUTSes. Sets of 100,000 usage records are used, and the share of matching usage records is either 1%, 10%, or 100%.

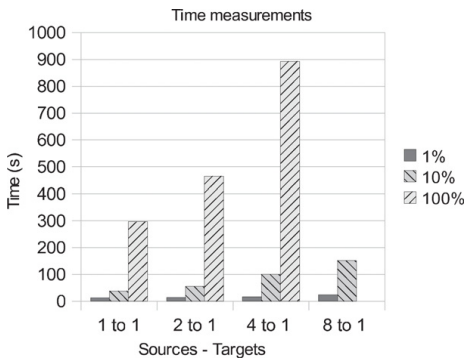


Fig. 10. In this scenario, usage records are transferred from several sources to a single target LUTS. Sets of 100,000 usage records with either 1%, 10%, or 100% of the records matching the specified query are also used in this scenario. A memory limitation prevents testing the case where a total of 800,000 records are sent from all eight sources to a single target LUTS.

measurements between a single source and a single target are included both in this scenario and in the previous one. As can be seen in the figure, the results are approximately linear with respect to the number of sources. As previously mentioned, reading the records consumes less time than writing the records, and so writing to the target becomes the limiting factor in this scenario.

The case where a total of 800,000 records are sent from eight different sources to a single target could not be completed due to a memory limitation of the Java container for the target LUTS in our test environment. The maximum heap size in Java on 32-bit Linux is roughly 1.7 GB [20], and very large amounts of data inserted into the SGAS database back-end over a short period of time will cause the virtual machine to run out of memory, as memory cannot be freed quickly enough by the Java Garbage Collector. This limitation can be avoided simply by performing the sharing of usage records more often, since this keeps the memory requirements down, but this is not interesting with regards to this performance evaluation as this in effect already is covered by the four-to-one usage scenario, running twice as often.

To put the measurements presented in Fig. 10 in perspective, the time to perform only writes to a target LUTS from several other computers was also measured. These measurements were performed by isolating the write calls in the LUTSfed component,

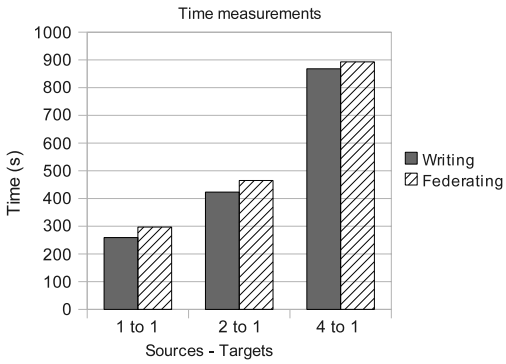


Fig. 11. The time of the total sharing process compared to the time performing only inserts of generated usage records in a remote LUTS. The time measurements are for 100,000 records per source, both for sharing and for pure inserts.

using data generated dynamically to avoid any IO overhead. These measurements are illustrated in Fig. 11.

From these figures combined, we can conclude that the usage record sharing does not infer a substantial overhead compared to doing writes directly to the database. The average delay of sharing compared to pure write operations is 150 μs, which is about 6.5% of the average total processing time. Because the difference between sharing records and pure write operations is so small, we can conclude that reading from the source LUTS and publishing to the target LUTS has been successfully parallelized. We can clearly see that the one-to-many scenario scales at least linearly with the number of targets, due to data caching in the source. In the many-to-one case, the times for sharing also scales linearly with regard to the amount of sources. In the worst case scenario (four-to-one), records were shared at a rate of approximately 2.2 ms per record, or just below 450 records per second. Notably, the ability to transfer 450 records per second is sufficient for most realistic usage scenarios today.

As an extreme example, let us consider a scenario where we need to transfer all data for a federated Grid from several distributed LUTSes into a single LUTS using a single LUTSfed component, and that the federated Grid includes one million compute nodes running jobs with an average duration of one hour. In this case, the single LUTSfed component would have to run for 37 minutes every hour. As the sharing is initiated and controlled by the LUTSfed component, rather than by each compute node, the gathering of records can be done in sequence without risk of congestion in case all nodes would happen to finish their jobs at the same time. If such or even more extreme use is anticipated, we would argue that a scalable deployment using multiple LUTSes and LUTSfed components would be a more normal configuration of the proposed architecture. As a matter of fact, with current technology, the ability to efficiently manage, aggregate, and in other way make efficient use of millions of user records per hour is more of a challenge than to make the proposed shared logging architecture scale to these numbers.

Comparing the measurements between, e.g., running the eight-to-one scenario with 10% matching records (for a total of 80,000 records in this case) and simply writing the equivalent of 100,000 matching records to the LUTS yields some interesting results. In this case, running eight-to-one is considerably faster even though the amount of data to transfer is only reduced by 20%. This is because when a single thread is doing only writes to the database (as in the test conducted) a new call to the target LUTS has to be done per part of the data set. If there are multiple threads writing to a LUTS at the same time, processing an incoming call to the service

in the front-end of the LUTS can be performed in parallel to a write operation in the database located in the back-end. This means that several concurrent write operation will be faster than a sequence of operations managed by a single thread. The same behavior can also be observed in the one-to-many scenarios, comparing, e.g., the time to write 10% matching records to eight different target LUTSes compared to writing 100% matching records once to a single LUTS. The differences are even more clear in this case, since the one-to-many scenarios can rely extensively on the database cache to reduce the effects of read operations, while the call processing overhead stays the same.

We remark that the sharing of usage data is normally not a time critical process, and therefore is performed in batches at regular intervals, such as once per day. Therefore, even in today's largest Grids, the sharing of usage records in realistic scenarios will be performed in minutes, or even seconds.

The performance of the modified LUTS (see Section 6.1) was also compared to the previous version of SGAS. In this test, a set of 100,000 records as before was used, with 10% of the records matching the specified query. The total time for sharing these records was measured several times using both target LUTSes running SGAS version 2.0 and target LUTSes running the modified 2.2.0 version. No significant difference in performance based on the SGAS version could be observed, and this clearly motivates the choice of additional resilience offered by these improvements.

8. Related work

Shared logging is an increasingly more important problem for accounting in overlapping and federated Grid environments. Despite this fact, the problem has so far only been shown limited attention in the literature. This could partly be due to the substantial amounts of efforts still being devoted to developing appropriate accounting solutions within specific Grids, and partly because of the additional complexities introduced by Grids running different middlewares and accounting systems.

However, some related projects should be highlighted. For example, the DEISA project [21] has put efforts into accounting systems with sharing functionality similar to the one described in this paper, although low-level details of their solution are not presented [22].

With the aim to facilitate exchange of accounting data between Grids, the OMII-Europe [23] has been working on interoperability between different accounting systems by making them support the RUS interface [24].

Very much related to the problem of usage logging in federated Grids are the challenges involved with creation federations or collaborations that incorporates existing Grid solutions. One approach based on a common gateway component and peering of requests is presented in [25]. Another approach presented in [26] is to create a hierarchy of Grids by abstracting remote Grids and presenting them to the local system in the same way as a local resource. These approaches can both be seen as suitable usage scenarios for the work presented in this paper.

Other clearly related work includes a range of accounting systems, and the German D-Grid evaluation and comparison of seven such systems (APEL, DGAS, GASA, GRASP, GSAX, Nimrod/g, and SGAS) [6]. Notably, also the D-Grid efforts aim at an architecture involving multiple accounting systems.

9. Concluding remarks

We have proposed a solution to the problem of federated Grid accounting by evaluating and presenting an architecture for shared logging. The proposed architecture has a strong focus on being optional and non-intrusive for existing accounting system installations, while still offering extensibility and adaptability to support complex usage scenarios. We have also presented the implementation of a component for use in SGAS environments that has been used to evaluate the performance of the approach.

The resulting LUTSfed component is highly configurable, light-weight, and flexible enough to cover a wide range of usage scenarios. Based on the performance evaluation presented, we conclude that the performance of the new LUTSfed component is sufficient for not being a bottleneck in realistic scenarios also on the most large-scale Grids of today.

General improvements to SGAS have also been made during this research, and performance evaluations show that we achieve greater resilience and reliability at an insignificant cost in performance.

The future plans for the extended SGAS architecture include further research on how to achieve shared logging in a, in terms of middlewares and accounting systems, heterogeneous environment. Outside the scope of this research project is also the customization and adaption of the presented architecture for production use. The use of the federated logging approach to provide usage data for performing Grid-wide fairshare scheduling [27] will also be investigated.

Acknowledgements

This work has been supported by the Swedish Research Council (VR) under contract 621-2005-3667, and by the RESERVOIR project supported by the European Community's Seventh Framework Program under grant agreement no. 215605. This research was conducted using the resources of High Performance Computing Center North (HPC2N).

We are grateful to Josva Kleist, Lars Larsson, Mats Nylén, Henrik Thostrup Jensen, Johan Tordsson, and Mattias Wadenstein for providing feedback on, and improving the quality of this work.

We are also grateful to the anonymous referees for their valuable comments and suggestions.

References

[1] LHC project webpage, December 2009. http://lhc.web.cern.ch/lhc/.
[2] P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, T. Sandholm, Scalable Grid-wide capacity allocation with the SweGrid Accounting System (SGAS), Concurrency and Computation: Practice and Experience 20 (18) (2008) 2089–2122.
[3] T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, A service-oriented approach to enforce Grid resource allocations, International Journal of Cooperative Information Systems 15 (3) (2006) 439–459.
[4] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, F. Galán, The RESERVOIR model and architecture for open federated cloud computing, IBM Journal of Research and Development 53 (4) (2009) Paper 4.
[5] E. Elmroth, P. Gardfjäll, O. Mulmo, T. Sandholm, An OGSA-based bank service for grid accounting systems, in: J. Dongarra, et al. (Eds.), Applied Parallel Computing. State-of-the-art in Scientific Computing, in: Lecture Notes in Computer Science, vol. 3732, Springer-Verlag, 2005, pp. 1051–1060.
[6] C.-P. Rückemann, W. Müller, G. von Voigt, Comparison of grid accounting concepts for D-Grid, in: Proceedings of the Cracow Grid Workshop, CGW'06, Academic Computer Centre CYFRONET AGH, Cracow, Poland. October 15–18, 2006.
[7] M. Göhner, M. Waldburger, F. Gubler, G.D. Rodosek, B. Stiller, An accounting model for dynamic virtual organizations, Journal of Grid Computing 7 (2) (2009) 181–204.
[8] I. Foster, Globus toolkit version 4: Software for service-oriented systems, Journal of Computer Science and Technology 21 (4) (2006) 513–520.
[9] R. Mach, R. Lepro-Metz, B.A. Hamilton, S. Jackson, L. McGinnis, Usage record format recommendation, Draft Rec-UR-Usage, Global grid forum, Usage record WG, March, 2005.
[10] Open grid forum, December 2009. http://www.ogf.org/.
[11] Nordic data grid facility, December 2009. http://www.ndgf.org/.
[12] SweGrid, December 2009. http://www.swegrid.se/.
[13] NorGrid, December 2009. http://www.norgrid.no/.
[14] National institute of standards and technology, systems and network security group, Draft NIST working definition of cloud computing, 2009.

[15] E. Elmroth, F. Galán, D. Henriksson, D. Perales, Accounting and billing for federated cloud infrastructures, in: 2009 Eighth International Conference on Grid and Cooperative Computing, GCC 2009, IEEE Computer Society Press, ISBN: 978-0-7695-3766-5, 2009, pp. 268–275.
[16] D. Winer, RSS 2.0 specification, 2003, http://cyber.law.harvard.edu/rss/rss.html.
[17] Feed informer, December 2009. http://feed.informer.com.
[18] A. Gupta, I.S. Mumick, Maintenance of materialized views: Problems, techniques, and applications, Bulletin of the Technical Committee on Data Engineering 18 (1995) 3–18.
[19] J. Clark, S. DeRose, et al., XML Path Language (XPath) Version 1.0, W3C Recommendation 16 (1999) 1999. http://www.w3.org/TR/1999/REC-xpath-19991116/.
[20] E. Pasch, Linux on system z performance hints & tips, http://www.linuxvm.org/present/SHARE111/S2591ep.pdf.
[21] Distributed European infrastructure for supercomputing applications web page, December 2009. http://www.deisa.eu/.
[22] J. Reetz, T. Soddemann, B. Heupers, J. Wolfrat, Accounting facilities in the European supercomputing Grid DEISA. 2007.
[23] OMII-Europe project page, December 2009. http://omii-europe.org/.
[24] S. Newhouse, J. MacLaren, Resource usage service RUS. global grid forum resource usage service working group draft-ggf-rus-service-4, 2005.
[25] M.D. De Assunção, R. Buyya, S. Venugopal, InterGrid: A case for internetworking islands of Grids, Concurrency and Computation: Practice and Experience (CCPE) 20 (8) (2008) 997–1024.
[26] E. Huedo, R.S. Montero, I.M. Llorente, A recursive architecture for hierarchical grid resource management, Future Generation Computer Systems 25 (4) (2009) 401–405.
[27] E. Elmroth, P. Gardfjäll, Design and evaluation of a decentralized system for grid-wide fairshare scheduling, in: e-Science, IEEE Computer Society, 2005, pp. 221–229.

Erik Elmroth is Professor, Head of the Department of Computing Science, and Deputy Director for the High Performance Computing Center North (HPC2N), at Umeå University, Sweden. His research background includes grid computing, parallel computing, algorithms for managing memory hierarchies, linear algebra library software, and ill-posed eigenvalue problems. He is co-recipient of the SIAM Linear Algebra Prize 2000, for the most outstanding linear algebra publication world-wide during the preceding three-year period. He currently leads the Grid computing research (www.gird.se) at Umeå University, focusing on infrastructure and application tools for grid and federated cloud computing. International experiences include a year at NERSC, Lawrence Berkeley National Laboratory, University of California, Berkeley, and one semester at the Massachusetts Institute of Technology (MIT), Cambridge, MA. Erik is member of the Swedish Research Council's Committee for Research Infrastructures (KFI) and Vice Chair of its expert group on e-science. He has been appointed the Scientific Secretary for two e-science groups of the Nordic Council of Ministers (NCM).

Daniel Henriksson is a Ph.D. student at the Department of Computing Science, Umeå University, Sweden. He received his masters degree in Computing Science in 2007, and began a Ph.D. program shortly after finishing his masters degree. His research areas are Grid and Cloud computing with topics such as accounting, federated Grids and Clouds, and elastic Cloud infrastructures.

II

Paper II

Accounting and Billing for Federated Cloud Infrastructures*

Erik Elmroth[1], Fermín Galán Márquez[2], Daniel Henriksson[1], and David Perales Ferrera[2]

[1] *Dept. Computing Science and HPC2N, Umeå University*
SE-901 87 Umeå, Sweden
{elmroth, danielh}@cs.umu.se
http://www.cs.umu.se/ds

[2] *Telefónica Investigación y Desarrollo, Spain*
{fermin, perales}@tid.es
http://www.tid.es

Abstract: Emerging Cloud computing infrastructures provide computing resources on demand based on postpaid principles. For example, the RESERVOIR project develops an infrastructure capable of delivering elastic capacity that can automatically be increased or decreased in order to cost-efficiently fulfill established Service Level Agreements. This infrastructure also makes it possible for a data center to extend its total capacity by subcontracting additional resources from collaborating data centers, making the infrastructure a federation of Clouds. For accounting and billing, such infrastructures call for novel approaches to perform accounting for capacity that varies over time and for services (or more precisely virtual machines) that migrate between physical machines or even between data centers. For billing, needs arise for new approaches to simultaneously manage postpaid and prepaid payment schemes for capacity that varies over time in response to user needs. In this paper, we outline usage scenarios and a set of requirements for such infrastructures, and propose an accounting and billing architecture to be used within RESERVOIR. Even though the primary focus for this architecture is accounting and billing between resource consumers and infrastructure provides, future support for inter-site billing is also taken into account.

* By permission of IEEE Computer Society Press.

Accounting and Billing for Federated Cloud Infrastructures

Erik Elmroth*, Fermín Galán Márquez†, Daniel Henriksson*, and David Perales Ferrera†
*Department of Computing Science and HPC2N, Umeå University, Sweden
Email: {elmroth, danielh}@cs.umu.se
†Telefónica Investigación y Desarrollo, Spain
Email: {fermin, perales}@tid.es

***Abstract*—Emerging Cloud computing infrastructures provide computing resources on demand based on postpaid principles. For example, the RESERVOIR project develops an infrastructure capable of delivering elastic capacity that can automatically be increased or decreased in order to cost-efficiently fulfill established Service Level Agreements. This infrastructure also makes it possible for a data center to extend its total capacity by subcontracting additional resources from collaborating data centers, making the infrastructure a federation of Clouds.**

For accounting and billing, such infrastructures call for novel approaches to perform accounting for capacity that varies over time and for services (or more precisely virtual machines) that migrate between physical machines or even between data centers. For billing, needs arise for new approaches to simultaneously manage postpaid and prepaid payment schemes for capacity that varies over time in response to user needs.

In this paper, we outline usage scenarios and a set of requirements for such infrastructures, and propose an accounting and billing architecture to be used within RESERVOIR. Even though the primary focus for this architecture is accounting and billing between resource consumers and infrastructure provides, future support for inter-site billing is also taken into account.

I. Introduction

Cloud computing has become an established paradigm for running services on external infrastructure, where virtually unlimited capacity can be dynamically allocated to suit the current needs of customers and where new instances of a service can be deployed within a short time frame. Although the term Cloud computing has come to include several kinds of technologies offering remote execution and service management, it is used in this paper to denote scalable elastic data center infrastructures offering dynamic and cost-efficient service provisioning.

There are many different Cloud computing solutions available, such as Amazon Elastic Compute Cloud [1]. However, different Cloud computing solutions are rarely compatible with each other and this creates a kind of vendor lock-in which is not only limiting to the customer, but also limits the potential of Cloud computing as a whole since separate Cloud computing solutions are unable to interoperate.

Grid computing can be seen as one of several predecessors to Cloud computing. Grid computing is often about making large computations using large amounts of resources, whileas Cloud computing is more about making large amounts of resources available to many different applications over a longer period of time. Clouds leverage modern technologies such as virtualization to provide the infrastructure needed to deploy services as utilities. Still, Cloud computing and Grid computing share a lot of the underlying technology and many concepts from Grid computing can be modified and made suitable for Cloud computing as well.

Resources and Services Virtualization without Barriers (RESERVOIR) [2] is a research project partly funded by the European Union, focused on federation of Clouds at the infrastructural level. The federated Cloud approach, where a single entity serves as a gateway to different independent solutions, is one way to solve the limited interoperability, as different technologies can be unified and abstracted towards the consumers. This approach is also a cost-efficient alternative to over dimensioning the amount of servers in order to cope with peak loads, as extra resources from other sites in the federated Cloud can be utilized during peaks. Similarly, underutilized resources can be made available for other sites during periods of lower load as an extra source of income.

There are two major challenges with regard to accounting and billing in federated Cloud infrastructures. Accounting and billing must be carried out in a fair and standardized way both: (a) between the user[1] and the infrastructure owner; and (b) between the sites making up the federation. In this paper, we focus on accounting and billing between the owner of the infrastructure and the consumer. Future support for inter-site accounting and billing is also taken into account and briefly mentioned, but most of the details are left for future work.

The main contribution of this paper is a proposal for a federated Cloud accounting and billing architecture primarily for use within the RESERVOIR project. The proposed architecture is motivated by usage scenarios and requirements, and also supplemented with a requirement fulfillment analysis to show how the architecture meets the requirements. Existing Grid accounting systems has been analyzed, and even though no existing system fulfills all requirements, the solution can be based on an existing Grid accounting system which is considerably modified and extended to provide the additional functionality.

The parts of the overall RESERVOIR architecture that are relevant for accounting and billing are described in detail in this

[1]In this paper, *user* and *customer* refer to the Service Provider (SP) that uses the cloud infrastructure to deploy services. The terms should not be confused with service *end users*, which could be customer from the point of view of the SP, but not from the point of view of the cloud infrastructure provider.

document, and more information on the RESERVOIR model and architecture can be found in [2].

The paper is organized as follows: Section II presents background information and a motivation of the work. Section III contains usage scenarios and requirements with regards to accounting in a federated Cloud environment. Section IV presents a summary of the analysis of existing Grid accounting systems, including brief descriptions of the different technologies. Accounting and billing for the RESERVOIR project is presented in Section V. Future work and some concluding remarks are given in Section VI.

II. Background and Motivation

In the context of RESERVOIR, and this paper, the term Virtual Execution Environment (VEE) is used to denote the isolated environment where customer applications are executed and maintained. This includes both Virtual Machines (VMs) and Virtual Java Service Containers (VJSC). VMs are managed using traditional virtualization technologies. VJSC is a technology currently developed by Sun within the RESERVOIR project, where Java applications can be deployed in virtual containers that can be handled similarly to virtual machines and thus migrated across hosts. The VEE concept offers advantages in isolation since each VEE contains one self-contained service component, in billing since the VEEs are well defined accounting units, and also in dependency management as constraints such as affinity can be realized between sets of VEEs. A *service* consists of one or several VEEs, and as explained later the number of VEEs in a service can change dynamically during the lifetime of the service.

A. RESERVOIR

RESERVOIR is a European Framework Programme project focused on creating an infrastructure for federated Clouds. An Infrastructure Provider (IP) is an organization operating one or more sites (i.e. data centers) in the RESERVOIR cloud, and the different IPs share load according to framework agreements among them. The IP normally does not interact directly with end users, but with a Service Provider (SP) that deploys services to the infrastructure to be used by the end users. Each SP may offer different business solutions and alternatives towards the end users depending on the needs of their customers, and any services can be hosted on the same IP.

As an IP technology, RESERVOIR incorporates Business Service Management (BSM) and also strongly advocates interoperability among Cloud providers [2]. One key aspect of BSM in the context of Cloud computing is dealing with Service Level Agreements (SLAs). These agreements can be seen as a mutual contract between the SP and the IP, regulating the expected allocated capacity per service that the SP should obtain for the agreed price and also any compensations for not fulfilling this agreement.

The RESERVOIR architecture, as illustrated in Figure 1, is a three-tier software stack where each layer has a clear and well isolated responsibility. The layers are separated by general interfaces designed to promote interoperability both horizontally between different Cloud providers, but also vertically between

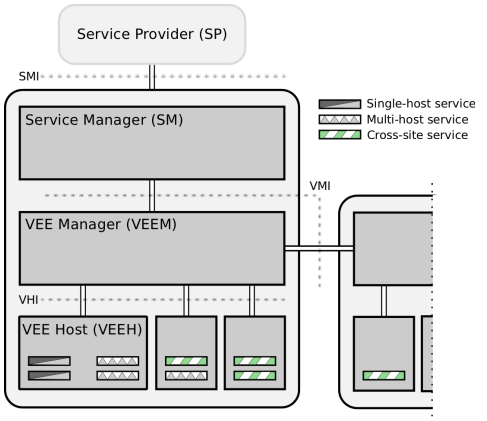


Fig. 1. The RESERVOIR architecture is made up of three different layers. The SM is the topmost layer, responsible for communicating with Service Providers and managing services on a larger scale. The VEEM layer is responsible for more fine-grained control over each service component, including placement and placement policies both locally and across sites. The VEEH layer hosts the VEEs and manages capacity allocation and metering. Well defined interfaces separates the layers, and the same interface (VMI) is used both between the SM and the VEEM and between the VEEMs of different sites. VEEs making up a single service can run either on a single host, on different hosts at the same site, or at different hosts belonging to different sites.

different implementations of each layer. There are three main interfaces, the VEE Host Interface (VHI) that separates the two lower layers, the VEE Manager Interface (VMI) that both separates the upper two layers and also is used for inter-site communication, and the Service Manager Interface (SMI) that provides service management functionality between the SP and the SM, and therefore is the primary interface between SPs and the RESERVOIR federated cloud.

The different layers of the RESERVOIR architecture are:

- **Service Manager:** The topmost layer of the architecture is the Service Manager (SM). Components at this layer are concerned with services as a whole rather than the specific VEEs that make up a service. This includes, e.g., accounting and billing, SLA enforcement, monitoring, and services deployment. Notably, the SM components are not aware of, or concerned with, where the VEEs making up a service are actually run. The interaction between the RESERVOIR infrastructure and the SPs is also handled at this layer.
- **VEE Manager:** Components at the VEE Manager (VEEM) layer are responsible for interacting with the SM and VEEH at the local site, but also horizontally with the VEEMs of other sites. Components at the VEEM layer are not concerned with services per se, but deals with sets of one or more VEEs that may have placement constraints (such as affinity) with other VEEs. The main responsibility of the VEEM layer is to optimize and manage the placement of VEEs, both locally and between different sites.
- **VEE Host:** The VEE Hosts (VEEHs) are responsible for

running and monitoring each single VEE. Each VEEH runs a specific virtualization technology, and translates commands sent by the VEEM through the common VEEH Interface to commands suitable for the underlying technology.

One important aspect of the RESERVOIR architecture is that a VEE can potentially run on one of several local hosts or even at a remote site. Also illustrated in Figure 1, the components on the SM level will never be aware of such placement decisions, as this placement is managed on, and abstracted by, the VEEM layer. It is also possible for a VEEM to re-locate running VEEs dynamically during the lifetime of the service. Similarly, the VEE itself is not aware of its placement (which can be remote or local relative to its origin), and this loose coupling between the VEE and the executing site is both a key feature and a complicating factor for, e.g., accounting and billing.

During inter-site communication, the VEEM of the local site will act as an SM with regards to the VEEM of the remote site. Since the same interface is used for inter-site and SM-VEEM communication, the same operations are used in both scenarios.

Monitoring and accounting data in RESERVOIR are made up of two different kind of measurements:

- The virtualization platform monitors the consumption and allocation of physical resources for each VEE.
- The disk images making up each VEE may contain special software that measures Key Performance Indicators (KPIs) that are used to provide application specific measurements. This makes it possible to formulate SLAs in application specific terms, e.g., the maximum number of active customers per server instance for a Web shop.

These data are processed by components in the SM both to identify SLA violations and perform billing. Two different kinds of payment model will initially be available:

- **Postpaid:** The SP is billed at regular intervals for the accumulated resource consumption during the previous billing period.
- **Prepaid:** Using this payment model, credits are purchased by the SP in advance and subsequently consumed in accordance with the resource consumption of the deployed services.

These payment models are analogous to models that have proven to be successful within, e.g., the mobile phone industry. By extending these models to also support compensations for SLA violations, the payment models should be able to fulfill the needs of RESERVOIR, while still being easy to comprehend.

III. Usage Scenarios and Requirement Analysis

An accounting and billing architecture for federated Cloud infrastructure has to be designed to meet the requirements of scenarios that are not present in traditional Grid (and possibly Cloud) infrastructures. Two identified challenges that strongly affect the design of the accounting and billing system are presented in the following sections, with each section covering one specific usage scenario.

A. Accounting for executing processes with unknown and dynamic placement

In a federated Cloud environment, the actual placement of the VEE will not always be known to the entire system, and may also change during the course of the service lifetime. This is especially true for RESERVOIR, due to the aforementioned abstraction of placement towards components in the SM.

Consider the case where a service running on RESERVOIR is made up of a single VEE. The deployment of this VEE is initiated by the SM. An identifier for the VEE that can be used to control the life cycle of the VEE is obtained upon deployment, but where the VEE is actually running is unknown to the components in the SM. During the course of the service execution, the placement of the VEE is subject to re-evaluation using placement heuristics by components at the VEEM layer, and this may result in the VEE being temporarily suspended and subsequently re-deployed either on another local host or on a remote host without notifying any of the SM components.

B. Accounting for services composed of an varying number of VEEs

An important concept of Cloud computing is that the number of VEEs composing a service can be dynamically increased or decreased to cope with a change in demand. A reduced demand implies that one or more VEEs can be shut down or suspended to make capacity available for other tasks and reduce costs for the SP, while an increase in demand can lead to new VEEs being deployed to share the load of the entire service.

In this scenario it is also possible that the accounting and billing configuration for the entire service is changed while the system is running. This could be, e.g., that the payment model is changed from prepaid to postpaid, and this must be possible to do without having to stop and re-deploy any running software.

C. Requirements

Sections III-A and III-B outline usage scenarios with challenges that are typical for federated Clouds, and for RESERVOIR in particular. A list of requirements have been extracted from these challenges. Furthermore, this list of requirements is complemented with general requirements that are not due to any of the two usage scenarios, but still are important to consider when designing the architecture.

- Req. 1 LOCATION_UNAWARENESS: The accounting system must be able to account for both VEEs running locally and those running at remote sites without being aware of the placement of any service component. This includes being adaptable to dynamic changes in placement of a VEE during its execution. Also relevant is that since a VEE is not aware of its own placement, it has to have a loose connection (if any) to the accounting system.
- Req. 2 SERVICE_ELASTICITY: The number of VEEs underlying and fulfilling a service can change dynamically, and the accounting system must be designed to handle this without relying on keeping track of the amount and identities of currently active VEEs.

- Req. 3 SERVICE_BILLING: The billing for the execution of a service must be done on a per service basis, and not for each VEE. Multiple billing methods must be supported (including, e.g., postpaid and prepaid). The payment model (or account) of a service must be changeable without affecting components outside the accounting and billing system, and without enforcing any restarts or redeployments.
- Req. 4 COMPLEX_PRICING: The function that calculate prices from the accounting information must be able to incorporate complex pricing rules depending of several factors such as, e.g., customer history or seasonal discounts.
- Req. 5 ADAPTABLE_DESIGN: The accounting system must be open for modifications to cope with future changes and enhancements. In other words, it must be possible to change the functionality in practically any component of the system to, e.g., add new hardware measurements and KPIs or change the interaction with other components.
- Req. 6 FLEXIBLE_DATA_FORMAT: The format used for accounting data must handle both hardware measurements, such as CPU or memory consumption, and any application specific KPIs (e.g. database transactions per second). It should also be possible to add aggregation functionality in the future, without modifying the format of the accounting data.
- Req. 7 SERVICE_ACCOUNTING: The accounting system must be suitable for long running services, and not limited to tasks with a limited execution time. It is also important that the solution supports aborting or suspending a running service due to, e.g., a lack of credits.
- Req. 8 COMPENSATIONS: Both usage and compensations (due to breaking SLAs) must be accounted. The system must support complex schemes incorporating arbitrary functions for calculating the compensations.

IV. Evaluation of Existing Grid Accounting Systems

As no accounting system focused on Cloud computing could be found, several different Grid accounting systems were evaluated as a first step to creating the accounting and billing architecture for RESERVOIR. By extending an existing Grid accounting system, less time has to be spent on the common mechanisms and the focus can instead be on developing the Cloud (and federated Cloud) specific functionality. Both commercial and open-source alternatives were considered, although commercial system are considerably harder to extend due to the closed source code. The full evaluation is out of scope for this paper, and only the result of the evaluation and brief information on the most promising candidates are presented herein.

The analysis of open source alternatives was focused on the Distributed Grid Accounting System (DGAS) [3], GridBank / Grid Accounting Services Architecture (GASA) [4], and SGAS [5]. Although several different commercial systems were considered, including Sun ARCo [6], HP Enterprise Usage Management [7], Verizon UMS [8], BMC Software Usage [9], and IBM Tivoli Usage and Accounting Manager [10] (ITUAM), we only present the open source alternatives in this paper. This is because detailed technical information to be used in a fair evaluation is not as easily available for the commercial solutions. For the open source alternatives, the in-depth comparisons done by [11] and [12] were used as reference.

A. Grid Accounting Systems

This section presents brief descriptions of the different Grid accounting system that proved the most likely candidates to be extended into the RESERVOIR accounting and billing system.

1) SGAS: SGAS is a capacity allocation system for Grid environments. A credit-based allocation model is used where projects are granted allowances to be spent across resources on the Grid. These allowances are collectively enforced by the resources in real-time, with the details of the enforcement specified by local policies. SGAS consists of three self-sustaining components, each responsible for a distinct and optional part of the accounting procedure:

- **Bank:** The bank service manages quotas and resource consumption for all resources using it, facilitating coordinated quota enforcement on the Grid. Credits are allocated to users and then consumed each time the user runs a job on the Grid.
- **Logging and Usage Tracking Service (LUTS):** XML-based usage records for completed jobs are published by the Grid sites and stored in the LUTS through a Web service interface. The LUTS uses an XML database backend to store the usage records in a native format.
- **Job Account Reservation Manager (JARM):** The component acts as a bridge between local resources and the Grid-wide accounting context. Local job submissions are intercepted by the JARM, and sufficient credits are reserved in the Bank prior to job execution. Once a job has been completed, this reservation is resolved and the surplus of reserved credits (if any) is returned to the Bank account.

A typical usage scenario for an SGAS-equipped system starts with a job being submitted to a Grid job manager, for example a WS-GRAM [13] component or ARC [14] GridManager, on a resource. Before the job is forwarded to the local resource manager for execution, the JARM makes a reservation of credits in the Bank. The amount of credits reserved is typically based on a user-specified estimate of the maximum amount of resources required to complete the job. If there are sufficient credits in the Bank, the job is executed. Once the job has completed, sufficient credits to cover the definite amount of consumed resources is deducted from the previous reservation, and any remaining credits are return. At this stage, a usage record containing the details of the job execution is stored in the LUTS. Local policies decides if a job that consumes more than the estimated amount of resources should be aborted or allowed to complete.

A Grid system setup can use one or more of the SGAS components independently of each other. For instance, if real-time quota enforcement is not required, and the system can run without the Bank component.

2) Distributed Grid Accounting System: DGAS is an accounting system originally developed for the European DataGrid project, and subsequently adopted by the Enabling Grids for E-sciencE (EGEE) project for further development. DGAS is designed to support a full-stack solution from usage metering up to account balancing supported by economical models. DGAS is composed of three independent layers, each responsible for a well defined part of the accounting procedure.

- **Usage Metering:** The usage metering layer is responsible for composing usage records by parsing logs from underlying batch systems, and subsequently pass the usage records to the above accounting layer. The metering process in DGAS is designed to ensure that the metering data can be distinctly mapped back to the executing user, the active resource, and the job currently being run.
- **Usage Accounting:** DGAS uses components called Home Location Registers (HLR) to manage accounts associated with users or resources. The HLR components can be responsible for a subset of usage records, making the system scalable and resilient to single component failures.
- **Account Balancing and Resource Pricing:** Special components called Price Authorities (PAs) are responsible for the resource pricing. The PAs support several different pricing algorithms that can be dynamically linked by the PA, making it possible to support the resource owners different requirements.

DGAS can be seen as a zero-sum system of resource exchange, where credits spent on a job run on resources owned by another virtual organization in turn can be distributed among (and subsequently spent by) users of that virtual organization.

3) GridBank / Grid Accounting Services Architecture: GridBank is an infrastructure for accounting in computational Grids. One of the major differences between GridBank and other accounting system is the support provided for computational economy and service cost negotiation.

The infrastructure in GridBank is based on a central server, connecting producers and consumers of Grid resources. The accounts are maintained centrally, making it convenient to manage users within the system.

GridBank is developed as a part of the Gridbus [15] project, and based on the Globus Toolkit [16]. Most notably, the security framework is reused to provide secure sockets and single sign-on mechanisms.

When submitting a job the client negotiates the service cost per time unit and picks the most suitable Grid Service Provider to run the job. The client contacts the GridBank Server and, given that the client has sufficient funds to run the job, a GridCheque is issued. The GridCheque is sent along with the job when the job is submitted. During job execution, usage records related to the job and these records are used to redeem the payments using the GridCheque.

B. Discussion

The requirements listed in Section III-C were used as a base for evaluation, and many of the requirements are specific for a Cloud (or even federated Cloud) environment. As a consequence, neither of the examined existing accounting systems (that are primarily designed for Grid usage) fulfill all the requirements without modifications. However, SGAS proved to be the alternative that is closest to the envisioned solution, even though a large amount of the functionality is missing.

The main reason why SGAS is deemed the strongest candidate is that the software is open source and the components of SGAS are very isolated in their concern, which makes it possible to reuse only some of the existing components. Another advantage of this loose coupling is that fresh components can be developed as independent modules that are not tightly coupled with other components in the SGAS accounting system.

DGAS was another candidate that is also open source, in productional use by, e.g., the EGEE, and supports different kinds of accountable resources. The main drawback of DGAS is the tight integration with the workload management system, as this limits the potential of adaptability to suit the needs of RESERVOIR. The non-standard components also make partly adaptions of the system more cumbersome and time consuming.

GridBank has several advantages, especially a strong security framework and usage records being compliant with the OGF format recommendation. Its potential of being the base for the RESERVOIR accounting and billing framework is however limited by having components that are not conforming to standards (making any extensions GridBank specific), and also by a strict architectural subdivision of consumers and provides.

V. Architecture for Accounting and Billing

A description of the resulting architecture for accounting and billing in RESERVOIR is presented in this section followed by an analysis on how the suggested architecture fulfills the previously identified requirements.

A. Proposed Architecture

This section describes a proposed architecture that combines the know-how of existing accounting system with the requirements and features of the RESERVOIR infrastructure. The architecture composes an *Accounting layer*, focused on collecting and managing the data which components in the *Billing layer* use as their input. Also included in the architecture is a *Business layer* that forms the link between the technical system and the SPs in terms of, e.g., pricing, invoicing, and service management. This paper is mostly focused on the Accounting and Billing layers, but the Business layer is also covered to offer a complete picture of the system. An overview of the different layers and their components can be seen in Figure 2. Some of the components in the architecture (shown with dashed borders in the figure), notably the SLA Violation Assessment, the VEEM Accounting Manager, and the Service Life-cycle Manager are gateway components between the accounting system and other parts of the RESERVOIR architecture. Also shown in the figure is the Accounting Database (ADB) and the Business Information Database (BIDB). These components are not specific to the architecture, and can be realized using any available database technology.

One important aspect is that neither of the accounting or billing components are concerned with the form of the

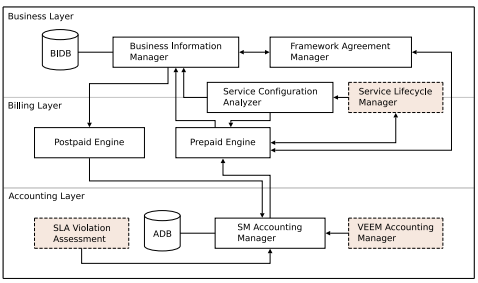


Fig. 2. Overview of the proposed architecture for accounting and billing within the RESERVOIR Service Manager. The figure shows the main components of the suggested architecture, with the connectors indicating the main interactions between components. Components with a dashed border in the figure handle the communication with other parts of the RESERVOIR SM system.

accounting data, apart from the presence of an ABC identifier (described in Section V-A2) and a site identifier which uniquely identifies the Infrastructure Provider where the accounted data was generated. The specific format and content of the accounting data are only the concern of the lower level components supplying the data and to the business components translating the data into credits. Thus, the attributes measured and billed for can change without affecting the accounting and billing components.

Accounting data used within RESERVOIR are based both on measurements on the system level and on the application level. On the system level, the VEEH can obtain information by measuring, e.g., the CPU or memory consumption of a VEE. On the application level, software specific KPIs are measured from inside the VEE using custom software (probes or agents), and the data are used to monitor special properties of the particular application.

The remainder of this section will discuss each layer in more detail, and describe the concern of the different components together with the main inter-component interactions.

1) Accounting Layer: The Accounting layer is responsible for the interaction with the surrounding infrastructure, collecting usage data from the VEEM level and data regarding SLA violations from other components in the Service Manager. The primary component is the *SM Accounting Manager* (SMAM) that together with the underlying Accounting Database (ADB) offers persistent storage and management of usage data and violations. The SMAM is supplied with data regarding SLA violations from the *SLA Violation Assessment* component, and these SLA violation are taken into account by components at the Billing layer. Similarly, the *VEEM Accounting Manager* (VAM) is responsible for collecting usage data from the local site, mark them with the site ID, and supply this to the SMAM at regular intervals.

The SMAM is the only component that interfaces with the ADB, and this single point of interaction makes it possible to abstract the technical details of the ADB from other components in the proposed architecture. This means the underlying database technology may be replaced without affecting any other component than the SMAM.

As mentioned in Section II-A, the VEEM acts as an SM towards a remote VEEM when dealing with migrated VEEs. This is also true when dealing with the accounting data, and the local VAM will act as an intermediate when dealing with accounting data received from the remote site (stamping the accounting data with its own site ID).

Aggregation and other kinds of data transformation can be added both to the VAM in order to reduce the network load between the SM and the VEEM, and to the SMAM to process the data before storing it in the database. Where to perform which kind of data transformation is specific for each site, and depends on, e.g., which interval is used for the accounting data and the capacity and size of the site in question.

2) Billing Layer: The Billing layer is primarily made up of the *Postpaid Engine* and the *Prepaid Engine*, supported by the Service Configuration Analyzer (SCA) and the Service Life-cycle Manager (SLM) that are part of both the business and billing layers.

When a service is deployed, the SLM contacts the SCA to validate the Deployment Descriptor (DD). This file is a description of the planned deployment of a service, including the hardware requirements. The SCA analyzes the DD from a business perspective to apply business oriented deployment restrictions. This could be, e.g., taking into account the customer history and the profitability of admitting the deployment. Included in the DD is also the payment method to use for the service, and the validity of these parameters, for instance that the user and account to be billed exists, is also established at this time. In the case of the prepaid payment alternative, the amount of available credits is also verified to ensure that the service is able to run for at least a short while before the credits run out.

When the planned deployment of the service has been verified, the SCA will generate a unique identifier for this particular service. This identifier is referred to as the Accounting, Billing, and Compensation (ABC) identifier, and all usage records and SLA violation reports concerning this service will contain this identifier. Using this approach, it is possible to address all VEEs of a service (or the usage data / violation data related to any VEE of a service), without knowing where they are run or how many they are. Similar results can be obtained by using the service identifier as the common parameter, and SCA implementations could use the service identifier used in others SM components as the ABC identifier. From the point of view of the accounting and billing architecture, the origin of the ABC identifier is not relevant, as far as each service has an unique ABC associated.

Note that all services, regardless of payment model, will have such an identifier. This makes it possible to change the payment model for a service dynamically, for example from postpaid to prepaid, without affecting any components outside the accounting and billing subsystem.

The SLM calls the SCA once to evaluate the deployment of the service (as mentioned above), and this call is followed by another when the service is actually deployed. This is because the time-span between admitting a service and actually deploying it can be very large (and there is formally no guarantee that an admitted service will ever be deployed).

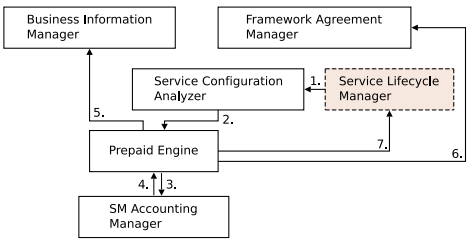


Fig. 3. The figure shows the procedure for the Prepaid Engine. The Prepaid Engine registers in the SM Accounting Manager to listen for updates when instructed to do so by the Service Configuration Analyzer. The Business Information Manager supplies the conversion from data concerning usage and violation to credits, and also makes decisions about what to do when the account runs low on credits.

The Postpaid Engine is responsible for generating an invoice from the data stored in the SMAM, when triggered by the Framework Agreement Manager (FAM) and Business Information Manager (BIM) components in the Business layer. These components are described in more detail in Section V-A3. This invoice generation can be triggered manually, but is typically done automatically at designated times to generate an invoice for, e.g., the previous month or quarter. When generating an invoice the BIM triggers the Postpaid Engine to gather the usage data from the SMAM, and also provides the Postpaid Engine with metadata required to convert the usage into credits. Also specified in this call is the period to bill, and which ABC identifiers to include in the invoice.

The procedure regarding prepaid accounts is described below, and also illustrated in Figure 3. When the deployment call (1) is made, the SCA will trigger the Prepaid Engine (2), and the Prepaid Engine will in turn contact the SMAM to start receiving updates for all records and violations containing the ABC identifier relevant for this service (3). The Prepaid Engine keeps track of the balance for all prepaid accounts in use, and makes the relevant withdrawals (or compensations) as usage data and violation records are received from the SMAM (4). The BIM supplies the Prepaid Engine with the mappings between the ABC identifiers and actual accounts to bill, and also supplies the pricing function. If an account is running low on credits, or when running out of credits, the BIM is contacted and the appropriate action is established based on business rules (5). When running low on credits, this could typically involve notifying the associated account owner by calling the FAM (6). When the credits are insufficient to continue running the service, a reaction could be, e.g., changing the payment model to postpaid, or instructing the SLM to suspend the running service (7).

The Prepaid Engine keeps the state of each account within the component itself in memory, and periodically sends a snapshot of the state to the BIM for persistent storage. If the Prepaid Engine is restarted, the previous stored state can be obtained from the BIM.

3) Business Layer: As briefly mentioned in the previous section, the main components of the Business layer are the BIM and the FAM. The BIM is responsible for storing and managing business information, based on an associated Business Information Database (BIDM). This information includes SPs, federated IPs (including the catalog of resources offered by each one) and the deployment restrictions used by the SCA. Specially relevant from the point of view of accounting and billing is the business context for the conversions from technical data to invoices or other business elements. These business contexts are supplied to both Payment Engines when generating bills or, in the case of the Prepaid Engine, converting from usage or violations to credits. This is also where the mapping between an ABC identifier and an account is made, and the persistence for accounts in the system is also within the purview of the BIM.

The FAM deals with user management including the external interface of the accounting and billing system. A user can access the FAM to, e.g., deposit credits, see the current balances of the user's accounts, browse the bills history, etc. The FAM can also notify the users when, for instance, an account is running out of credits. The FAM also offers interfaces for administration used to, e.g., manage user accounts or configure the resource catalog and pricing functions stored in the BIM.

B. Requirements fulfillment

In this section we analyze how the proposed architecture effectively addresses and fulfills the requirements enumerated in Section III-C.

- Req. 1 LOCATION_UNAWARENESS: Each VEEM Accounting Manager collects accounting data from all VEEs running at that particular site. This way, all accounting data originating from a VEE are sent to the VEEM currently responsible for the execution of the VEE. The data are then propagated backwards until they reach the VEEM Accounting Manager, and subsequently components in the SM, on the primary site (the site from which the VEE originates). This way, the SM does not have to be aware of where the VEE is deployed in order to collect the accounting data, and the VEE is not aware of the data being propagated to another site if the VEE is deployed remotely. Note that the presence of the site ID does not violate this requirement.
- Req. 2 SERVICE_ELASTICITY: As described in Section V-A2, each VEE has an associated ABC identifier that can be used to map between a VEE and the service to which it belongs. This way, the amount of VEEs making up a service can change dynamically without affecting the accounting and billing system.
- Req. 3 SERVICE_BILLING: This requirement is also fulfilled using the proposed ABC identifier. The mapping between a group of VEEs and the associated account can be changed by modifying the mapping within the accounting system. The identified account in turn determines which payment method is used. A single customer can have several accounts, where each account is of either payment type.
- Req. 4 COMPLEX_PRICING: The billing components are supplied with business context information from the BIM making the actual mapping between usage and credits

decoupled from the billing process itself. This means the complexity is managed within the business model, making it a policy decision rather than a mechanism.

- Req. 5 ADAPTABLE_DESIGN: Large parts of the accounting and billing system are developed specifically for this project, and the external components that are being incorporated into the design are available under compatible open source licenses. Since the source code is available, it is possibly to change the behavior of any component within the system as necessary.
- Req. 6 FLEXIBLE_DATA_FORMAT: The format of the accounting data is not relevant to neither the accounting nor the billing components (as explained in Section V-A). This means that the format of the accounting data can change, as long as the ABC identifier and site ID are present.
- Req. 7 SERVICE_ACCOUNTING: The accounting system does periodical measurements of usage, and so is not dependent on a service finishing executing within a designated time frame. In addition, the Prepaid Engine can contact the SLM to suspend or cancel a service (depending on local policies) when the account used by the service is running out of credits.
- Req. 8 COMPENSATIONS: From the point of view of the accounting and billing system, compensations are dealt with in the same way as accounting data, although the data is supplied by different components. The main reason for the separation of SLA violation detection and accounting data management is that several components, such as those managing deployment of new instances, might be affected by an SLA violation while the accounting data is only relevant for the accounting and billing subsystems. The business components are then responsible for converting data concerning both usage and SLA violations into credits, and the billing components treats this information in the same way when creating invoices or modifying the balance of a prepaid account.

VI. Conclusions and Future Work

In this document we have presented a solution for an accounting and billing architecture for use in RESERVOIR and possibly other federated Cloud environments. Although neither of the Grid accounting systems fulfills all the identified requirements for federated clouds, SGAS was found to be the most suitable one to be used as a starting point. Finally, the suggested approach was also evaluated with regard to the requirements to ensure that no known issues remain unresolved.

The focus of the work done so far has been the design of the accounting and billing system, taking existing alternatives, the overall RESERVOIR architecture, and the requirements of the solution into account. This has resulted in a loosely coupled architecture for accounting and billing that is also flexible enough to be adapted to future changes in requirements.

Some parts of the system are already under development, namely those concerning deployment (BIM and SCA) and the most central components for the accounting part (the accounting managers). Future work includes implementing and evaluating remaining parts of the system, and integrating all components with the general RESERVOIR architecture. In addition, inter-site accounting and billing are also yet to be fully integrated with the suggested architecture.

Acknowledgments

This work has been partly supported by the RESERVOIR project as a part of the European Community's Seventh Framework Programme under grant agreement no. 215605.

We also thank Lars Larsson, Johan Tordsson, and Emilio Torres for providing feedback on, and improving the quality of this work.

References

[1] Amazon Web Services, LLC., "Amazon Elastic Compute Cloud," Visisted March 30, 2009, 2006. [Online]. Available: http://aws.amazon.com/ec2/

[2] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, L. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán, "The RESERVOIR Model and Architecture for Open Federated Cloud Computing," *IBM Systems Journal*, 2009, to appear.

[3] R. Piro, A. Guarise, and A. Werbrouck, "An Economy-based Accounting Infrastructure for the DataGrid," in *Proceedings of the 4th International Workshop on Grid Computing (GRID2003)*, 2003.

[4] A. Barmouta and R. Buyya, "GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration," in *Workshop on Internet Computing and E-Commerce, Proceedings of the 17th Annual International Parallel and Distributed Processing Symposium (IPDPS 2003), IEEE Computer Society Press, USA, April*, 2003, pp. 22–26.

[5] P. Gardfjäll, E. Elmroth, L. Johnsson, O. Mulmo, and T. Sandholm, "Scalable Grid-wide capacity allocation with the SweGrid Accounting System (SGAS)," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 18, pp. 2089–2122, 2008.

[6] Sun Microsystems, "SGE Accounting and Reporting Console (ARCo)." [Online]. Available: http://wikis.sun.com/display/GridEngine/Accounting+and+Reporting+Console+(ARCo)

[7] HP, "Enterprise Usage Management Solution overview and features." [Online]. Available: http://h20229.www2.hp.com/products/ium_ent/index.html

[8] Verizon, "Usage Management System." [Online]. Available: http://www22.verizon.com/it/products-services/telecom-software-applications/billing/mediation-rating/ums-index.html

[9] BMC Software, "BMC Software Usage." [Online]. Available: http://www.bmc.com/products/proddocview/0,2832,19052_19429_18235405_106787,00.html

[10] IBM, "Ibm tivoli usage and accounting manager - product overview," Visited March 31, 2009. [Online]. Available: http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.ituam.doc_7.1/overview/tuam_pdf_overview.pdf

[11] M. Göhner, M. Waldburger, F. Gubler, G. Rodosek, and B. Stiller, "An Accounting Model for Dynamic Virtual Organizations," University of Zürich, Department of Informatics, Tech. Rep. No. 2006.11, November 2006.

[12] C.-P. Rückemann, W. Müller, and G. von Voigt, "Comparison of Grid Accounting Concepts for D-Grid," in *Proc. Cracow Grid Workshop 06, Cracow*, October 2006.

[13] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *IFIP International Conference on Network and Parallel Computing, LNCS 3779*, H. Jin *et al.*, Eds. Springer-Verlag, 2005, pp. 2–13.

[14] B. Konya, "Advanced Resource Connector (ARC)-The Grid Middleware of the NorduGrid," *Lecture Notes in Computer Science*, pp. 10–10, 2004.

[15] R. Buyya, "Grid Economy Comes of Age: Gridbus Technologies for Service-Oriented Cluster and Grid Computing," in *Proceedings of the 2 nd IEEE International Conference on Peer-to-Peer Computing (P2P 2002), Linkoping, Sweden, Sept*, 2002, pp. 5–7.

[16] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit. Intl J," *Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.

III

Paper III

Scheduling and Monitoring of Internally Structured Services in Cloud Federations*

Lars Larsson, Daniel Henriksson, and Erik Elmroth

Dept. Computing Science and HPC2N, Umeå University
SE-901 87 Umeå, Sweden
{larsson, danielh, elmroth}@cs.umu.se
http://www.cs.umu.se/ds

Abstract: Cloud infrastructure providers may form Cloud federations to cope with peaks in resource demand and to make large-scale service management simpler for service providers. To realize Cloud federations, a number of technical and managerial difficulties need to be solved. We present ongoing work addressing three related key management topics, namely, specification, scheduling, and monitoring of services. Service providers need to be able to influence how their resources are placed in Cloud federations, as federations may cross national borders or include companies in direct competition with the service provider. Based on related work in the RESERVOIR project, we propose a way to define service structure and placement restrictions using hierarchical directed acyclic graphs. We define a model for scheduling in Cloud federations that abides by the specified placement constraints and minimizes the risk of violating Service-Level Agreements. We present a heuristic that helps the model determine which virtual machines (VMs) are suitable candidates for migration. To aid the scheduler, and to provide unified data to service providers, we also propose a monitoring data distribution architecture that introduces cross-site compatibility by means of semantic metadata annotations.

* By permission of IEEE Computer Society Press.

Scheduling and Monitoring of Internally Structured Services in Cloud Federations

Lars Larsson, Daniel Henriksson, Erik Elmroth
Department of Computing Science and HPC2N
Umeå University, Umeå, Sweden
Email: {larsson, danielh, elmroth}@cs.umu.se

***Abstract*—Cloud infrastructure providers may form Cloud federations to cope with peaks in resource demand and to make large-scale service management simpler for service providers. To realize Cloud federations, a number of technical and managerial difficulties need to be solved. We present ongoing work addressing three related key management topics, namely, specification, scheduling, and monitoring of services. Service providers need to be able to influence how their resources are placed in Cloud federations, as federations may cross national borders or include companies in direct competition with the service provider. Based on related work in the RESERVOIR project, we propose a way to define service structure and placement restrictions using hierarchical directed acyclic graphs. We define a model for scheduling in Cloud federations that abides by the specified placement constraints and minimizes the risk of violating Service-Level Agreements. We present a heuristic that helps the model determine which virtual machines (VMs) are suitable candidates for migration. To aid the scheduler, and to provide unified data to service providers, we also propose a monitoring data distribution architecture that introduces cross-site compatibility by means of semantic metadata annotations.**

I. Introduction

Cloud computing has the potential to offer cost-efficient and seemingly unlimited computational capacity to resource consumers, and more importantly, to deal seamlessly with unexpected spikes in resource consumption that would be unmanageable for in-house hosting alternatives. The problem of maintaining sufficient resources is transferred from the resource consumers to Cloud *Infrastructure Providers* (IPs). We refer to the consumers of Cloud infrastructure as *Service Providers* (SPs), which typically are companies who in turn offer services to end users. *Service-Level Agreements* (SLAs) specify the terms under which the SP provisions resources from the IP and at what cost, and define economical penalties if the IP fails to deliver accordingly.

IPs can collaborate on workload sharing and resource subcontracting to easier cope with spikes in resource consumption or other unexpected events that affects hosting of services. Such collaboration may exploit pricing differences at Cloud IPs which can yield savings, even for a low amount of requested resources [1]. We use the same definition for *Cloud federations* and *framework agreements* as in [2], namely that Cloud federations allow IPs to subcontract resources at remote Cloud sites when local resources are running low, as governed by bilateral framework agreements. The SP needs not be aware of such subcontracting and only interacts with the original IP. *Cloud bursting* can be seen as a special case of federation where resources are only provisioned by one party from the other, usually by a private Cloud from a public provider. Alternatively, an SP may directly host a service across several IPs. As in [3], we refer to this as a *multi-provider hosting* and consider it to be separate from Cloud federations. In multi-provider hosting, management and service orchestration across several sites is managed by the SP. In Cloud federations, the IP manages provisioning and monitoring of remote resources on behalf of the SP. IP-level management of e.g. elasticity and SLAs in Cloud federations [4] or federation/multi-hosting hybrids [3] is currently under research.

In this paper, we present ongoing work related to solving core management issues that arise specifically in Cloud federations. Specifying service structure and placement constraints affords the SP a sufficient amount of control over service deployment in Cloud federations. Schedulers must take this information into account when determining placement for each service component, and may use migration as a tool to optimize placement according to some management objective. Once a component has been placed and is executing, its state must be monitored to make placement optimization possible. Our contributions are the following:

- we define a hierarchical graph structure for service representation and intra-service rule specification which impacts scheduling within the Cloud federation,
- we present a scheduling model and heuristic that optimizes VM placement via local and remote migration, and
- we present a semantic monitoring data distribution architecture, which provides interoperability between different Cloud infrastructure monitoring systems.

The remainder of the paper is organized as follows. Section II briefly describes the design principles and the features that motivate our work. Section III presents how a graph may be used to represent structured services with rules concerning component placement and includes an example thereof. In Section IV, we present a model and heuristic for a scheduler that takes placement constraints into account for local and remote placement of VMs in the Cloud federation. Section V introduces an architecture of a system aimed to provide compatibility for disparate monitoring systems via employing semantic metadata to bridge the differences. The paper is concluded in Section VII.

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

II. Design Principles and Motivating Features

In this section, we briefly describe the design principles and features that motivate our work. We formulate the principle of *location unawareness* based on [5] and [6] such that it states that *neither* the management system *nor* the VMs should be needlessly aware of current VM placement. From the management point of view, this means that e.g. the scheduler is perfectly aware of whether a given VM is placed at a local host or at a remote site R, but it does not know *which particular host* at R hosts the VM (and it cannot request to change this placement). The VM may even have been delegated to another partner site by R without notifying the original IP.

From the VM point of view, location unawareness implies that the VM is not aware of its current hosting within the Cloud federation, including its location in the network. Thus, virtualized overlay networks must span across sites and allow VMs to keep all private and public IP addresses, even during migration from one site to another. Offering such networking functionality is the topic of ongoing research [6] and currently not offered by any commercial vendors.

Data and computation provisioning in federated Clouds raises concerns regarding locality, both from a performance and a legislative point of view [7], [8]. To ensure that resources are provisioned satisfactorily while retaining location unawareness, *affinity* and *anti-affinity* rules may be specified. We use the same definition of affinity as [9] i.e. to denote a set of placement constraining relationships between sets of related VMs. We use the term *AA-constraints* where both affinity and anti-affinity are applicable, and each term alone if something applies only to either affinity or anti-affinity.

Without loss of generality, we consider three levels of *AA*-constraints, namely host, (Cloud) site, and geographical region. For an affinity level L, if VM types A and B are in the relation, a scheduler must place all instances of these types so that placement restrictions are adhered to, e.g. instances must be placed on the same host machine or at the same site if this is the specified affinity relation. Conversely, anti-affinity requires that instances of VM types may *not* be placed on the same level, e.g. on the same host or at the same site. Using several *AA*-constraints, it is possible to restrict placement such that, e.g., all VMs must be placed on different hosts, avoid a certain competitor site, and may never be migrated to or placed in a region where certain legislation applies.

III. Service Representation

Some model is required to allow the SP to specify both the structure of the service and *AA*-constraints. We propose that hierarchical directed acyclic graphs (DAGs) are suitable service representations. The reasons are twofold: (a) there is an implied or explicitly stated structure between resources, e.g. between attached storage units and computational resources (parent-child relationship); and (b) *AA*-constraints may apply only to certain related service subsets (sibling relationship). In our formulation, trees are insufficient since a node may require more than one parent, for example if a VM is part of two otherwise disjoint internal networks.

Table I
Node types used to define the structure of a service.

Node type	Abbr.	Description
Service Root		Common ancestor for all service components.
Compute Resource	C	Compute resource, which can be connected to networks and storage units.
AA-constraint	A	Metadata for use within a scheduler to determine placement according to affinity and anti-affinity rules. *Scope* may either be *type* or *instance* and must be specified.
Block Storage	S_b	A mountable data storage for a Compute resource. Cf. Amazon EBS.
File Storage	S_f	Data storage which may be accessed by multiple Compute resources simultaneously. Cf. Amazon S3.
Internal Network	N_i	Internal network for all underlying Compute resources and File storages.
External Network	N_e	External network connection (IP address) for the parent Compute or File storage resource.

Special meaning is reserved for the words *type* and *instance* when used to describe resources: types act as templates for instances, and one-to-many instances can be instantiated of each type. Table I lists node types with description and abbreviation. Nodes of type *AA*-constraints (A) only affect Compute (C) and File storage (S_f) nodes. The other resources, networks and block storage, implicitly or explicitly belong to instances of either C or S_f, and thus are covered by the same *AA*-constraints as the node to which they belong.

Figure 1 shows examples of structures which can be composed into valid hierarchical DAGs. The relationship marked with edges create parent-child relationships. Instances of child nodes are attached to each instance of their parent. Both A and internal network (N_i) nodes may be nested to arbitrary depth. Nodes of type A stipulate constraints for all descendants as described above. For nested N_i nodes, C and S_f nodes require a virtual network interface for each ancestor of type N_i and each descendant of external network (N_e) nodes to connect them to each of these network instances.

An *AA*-constraint affects all descending C and S_f nodes but may have different scope, either type or instance, as specified as an attribute of the constraint. An *AA*-constraint with *type scope* affects how instances of a type can be placed in relation to instances of *other* types, but not instances of the same type. An *AA*-constraint with *instance scope* affects all descending instances *regardless of type*, and therefore also affects instances of the same type. For example, consider an *AA*-constraint A_1 specifying "not same host" with two underlying compute node types C_1 and C_2:

1) If the scope of A_1 is *type scope*, no instance of type C_1 may be placed at the same host as an instance of type C_2. (However, two instances of C_1 may be placed at the same host.)
2) If the scope of A_1 is *instance scope*, no pair of instances of either type (C_1 or C_2) may be placed at the same host.

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

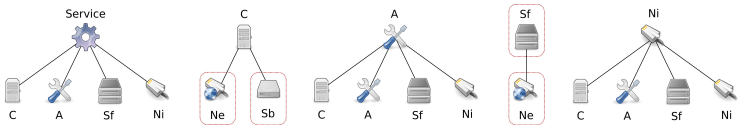


Figure 1. Rules defining valid inter-node relationships for service definition DAGs. C denotes Compute resources, A denotes *AA*-constraints, S_f and S_b denote file and block storage, respectively, and N_i and N_e denote internal and external networks. Valid terminal nodes are marked with a border. Further node description can be found in Table I.

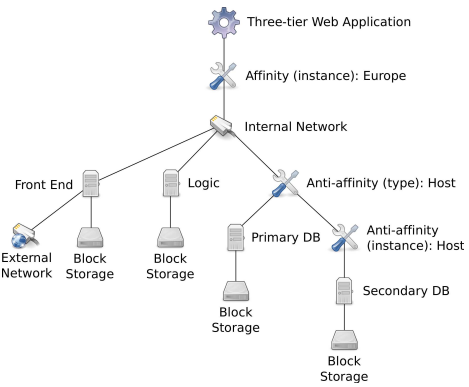


Figure 2. Example of a three-tier Web application service represented using a DAG which includes *AA*-constraints and network setup. Node types are shown in Figure 1, and labels have been added for clarity.

A. Service Definition Example

We exemplify this structure by describing a typical three-tier Web application in Figure 2 as a DAG. Immediately below the service root node, an *AA*-constraint states that all descendants of all resource types must be located in Europe. Thus, a scheduler may choose freely among Cloud federation partner sites located in Europe, but not elsewhere. An internal network resource node specifies that all its descendants are connected to a single local network instance. In addition, instances of the front end compute resource type are accessible via per-instance individual external IP addresses. A type scope anti-affinity constraint forbids placement of instances of the primary and secondary database servers at the same physical host. For the secondary database servers, an instance scope anti-affinity constraint explicitly forbids placement of instances at the same host, for fault-tolerance reasons. An individual block storage is attached to each compute node instance.

IV. Model for Scheduling in Federated Clouds

Scheduling is the process by which a VM management system decides on which physical host machine or partner site within a Cloud federation a VM should be placed. The general problem is to create a placement mapping between VMs and physical hosts such that placement fulfills certain management objectives [3], e.g. to maximize profit, avoiding loss of reputation, maximizing resource usage, etc. Mappings are evaluated using a number of factors, e.g. power consumption of physical host machines, economical penalties stipulated in pertinent SLAs, etc.

We present fundamental ongoing work for scheduling based on a model that takes *AA*-constraints, e.g. the ones shown in Figure 2, into account. The model assumes that migration can be used to optimize placement, but avoids unnecessary or risky (in terms of SLA violation risk) migrations.

The model regards remote sites as logical local hosts with different service-level characteristics, e.g. network capacity. Thus, management is simplified while still representing the performance and SLA-related differences between the local and the remote site(s).

Our model is formally described as follows. Let V be the set of VMs that need placement and H be the set of hosts to our disposal (including remote sites as logical members of H). M denotes a set of mappings $m_{v,h} \in M$ of VM v to host h stating that VM v is placed on h. Time is discretized and each interval has one active mapping. We wish to determine a new mapping M_n based on an old mapping M_{n-1} such that net profit is maximized. Net profit is expressed as the difference between a benefit function $B(V)$, a cost function $C(M)$ (models e.g. power usage due to the current host utilization), and estimated SLA-related costs due the inherent risk of performance loss associated with migration $S(M_{n-1}, M_n)$ in modifying the old mapping into the new one. We express this in Equation 1.

$$maximize\left(B(V) - \sum_{h=1}^{H}\sum_{v=1}^{V} C(M_n) - S(M_{n-1}, M_n)\right) \quad (1)$$

Note that if a mapping M makes use of remote resources in the Cloud federation, this will likely incur a larger cost $C(M)$ but (hopefully) also reduce the expenses if an SLA is violated, since the remote site also must provide compensations in that case. For sufficiently large problem instances, investigation of all possible new mappings to determine which gives sufficiently small values for $S(M_{n-1}, M_n)$ is too computationally intensive to be feasible. To that end, we define a heuristic to avoid wasting time investigating migrations that have a high risk of resulting in SLA violations.

A. Migratability heuristic

We define a *migratability* function $Mig(v, M)$ of a VM v given a current mapping M, where low migratability value implies that migration of v from its mapping in M is less

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

desirable. The scheduler uses this heuristic in an attempt of minimizing $S(M_{n-1}, M_n)$ from Equation 1, while still being open to performing migrations to optimize placement.

Due to affinity relationships, it is not sufficient to consider the migratability of a single VM in isolation. Rather, for a given proposed migration of a VM v from one host or site to another location, let O denote the set of other VMs that must also be migrated due to affinity constraints. We then define $Mig(O, M)$ as the migratability function for all $o \in O$, relative to the mapping M. Obviously, if the selected new location for a VM is a remote site, the scheduler uses site and geographical level affinity to determine eligibility since actual host deployment is not known at remote sites due to location unawareness. The remote site must abide by affinity rules or reject the request to run the VMs if unable to do so. Also, due to anti-affinity constraints, the set O may be limited in which host machines may be used for placement of the VMs. The value of $Mig(O, M)$ depends on the migratability value of each individual VM $o \in O$.

For a single VM v, the factors that determine $Mig(v, M)$ relate to the cost and risk of violating pertinent SLAs. The risk calculation is based on:

- Long-term high-level monitoring data collected on the usage patterns of the VM and the service it belongs to. For instance, this helps determine if the service usually peaks in usage at some regular intervals, e.g. the end of the month.
- Short-term low-level monitoring data from the hypervisor internals regarding the memory usage of the VM. As the number of dirtied memory pages per time unit increases, estimated migration time for the VM increases [10].
- Sizes of storage and volatile memory that have to be transferred to the new destination and current network utilization, as well as other currently active migrations. If shared storage is used, typically only volatile memory must be transferred. If, however, the VM is to be migrated to another Cloud, it may be required to transfer the regular storage as well.

The migratability heuristic prunes the search space and helps the scheduler concentrate only on potentially fruitful mappings. The heuristic identifies and confirms the intuition that the easiest VMs to migrate are the ones that have few affinity (and to lesser extent, anti-affinity) relations to other VMs, are not currently (or in the near foreseeable future) highly active, and such that decreased performance due to migration will not be costly in terms of SLA violations.

As summarized in [11], even in research VM management projects, schedulers are quite rudimentary: by default, only various subsets of greedy, round-robin, and explicit (manual) scheduling are supported. Most schedulers will also avoid performing migration of a VM once it has found its initial placement, which leads to sub-optimal performance and possibly higher energy costs than necessary. Although research has been made on this topic [2], there is to our knowledge currently no scheduler software that takes *AA*-constraints into account that is open to the research community.

V. Monitoring Data Distribution in Cloud Federations

All Cloud sites offer monitoring of virtual resources, however, there are many different and incompatible monitoring systems in current use and this causes integration problems. We present our ongoing MEDICI project, a monitoring data distribution architecture that collects data from various existing monitoring systems, marks it up with semantic metadata, and publishes it to subscribers, one of which is a semantic database. The database allows complex queries on the semantic self-describing data, and the result can be transformed into a desired output format.

The MEDICI architecture is designed to leverage existing software for its core operation in a scalable way. The components of the architecture shown in Figure 3 are as follows:

- *Monitored infrastructure*. A virtual Cloud infrastructure that is monitored continuously, e.g. computational resources, storage entities, and interconnecting networks.
- *Data annotator/publisher*. Data annotators and publishers are the core of the MEDICI system, providing:
  - Canonicalization and semantic annotation of monitoring values by plugins. The annotations conform to OWL (Web Ontology Language) ontologies, facilitating parsing and conversion at the consumer level.
  - Preparation of annotated monitoring data which is then published to the distribution hub.
- *Distribution hub*. Distribution hubs distribute semantically annotated monitoring data to a set of subscribers.
- *Subscribers*. Any consumer implementing the hub's protocol may be a subscriber, enabling e.g. external components, the SPs, and other Clouds in the federation to gain access to the data using a single hub. As shown in Figure 3, the hub may distribute both public and private streams of data. This distinction makes it possible to prevent inappropriate disclosure of data to different parties.
- *SPARQL endpoints*. SPARQL [12] endpoints are databases that act as subscribers and are deployed either locally or remotely. They make it possible to aggregate data from the federation and make SPARQL queries on the data.

The architectural components in MEDICI are designed to expose remotely invokable interfaces and the number of instances of each component may due to loose coupling be independently increased to handle scalability gracefully.

The raw monitoring values and basic metadata (interval length, information source, and monitoring system identifier for future parsing by plugins) are transferred from the monitored infrastructure to the data annotator/publisher using light-weight REST methodology by system-specific plugins. The data may be extracted using e.g. libvirt [13], which is compatible with several underlying hypervisor technologies. Plugins may also be developed for other monitoring systems, e.g. collectd and Nagios. Higher-level service-specific data, e.g. "number of currently logged in users", can also be distributed by the system.

The data annotator/publisher maintains a separate set of plugins for handling various input of raw monitoring values. Upon data arrival, the appropriate plugin creates a semantically

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

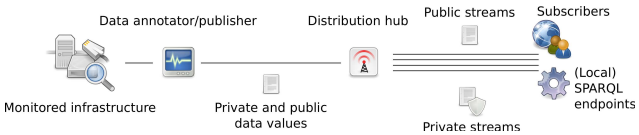


Figure 3. Overview of the MEDICI monitoring data distribution architecture.

annotated transformation from the raw data format in MEDICI canonical form. The data is then transferred to a distribution hub, which handles delivery to the subscribers.

The MEDICI canonical form for infrastructure data is based on the data set provided by libvirt. This choice was made for two reasons: (a) libvirt is compatible with most popular hypervisors; and (b) libvirt provides a reasonable subset of infrastructure-related measurements. However, note that since MEDICI uses extensible OWL ontologies, specific plugins can be developed for any input format. This allows service-specific data to be distributed.

The SPARQL endpoint acts as a subscriber to the hub and exposes its data via a rich semantic query language. This may be used for complicated queries, including queries for inclusion or inspection of remote monitoring data and accounting in a federated Cloud setting. It is i.e. possible to make queries that transform the remotely published data into data using the same measurement intervals as done locally, making it easier to apply the same mathematical functions for accounting and SLA violation detection purposes.

The distribution hub conforms to the PubSubHubbub [14] protocol, which uses the Atom format for data transport. We consider Atom suitable for this purpose for several reasons: (a) it is simple, incurs relatively low overhead, and is well-defined; (b) it is easily viewable in a Web browser or feed aggregator, requiring very little special software for a large variety of use cases; and (c) as an XML format it is easy to translate into other formats, and can transport other (semantic) XML data well, in addition to being platform independent. PubSubHubbub enables close to real-time updates of information in a scalable way, and by design of the PubSubHubbub protocol, the functionality of the hub is transparently hidden from consumers.

The strengths of this approach are that (a) plugins can be developed for specific monitoring already in use at Cloud sites; (b) plugins should not have a large negative performance impact on monitoring systems; and (c) publishing data to a database upon which semantic queries can be invoked, the data from a remote site can be queried and transformed into a format that is compatible with the monitoring system on the local site.

The architecture enables location unawareness from the management point of view, since it aids in bridging the gap between the management systems used at different Cloud sites, making monitoring data from one Cloud site easily integratable with the other.

VI. RELATED WORK

Service structure does currently not have wide-spread support. APIs such as Amazon EC2 or Open Cloud Computing Interface (OCCI) allow the SP to specify parent-child relationships (e.g. storage unit s is the child of VM v), but do not support sibling relationships such as the anti-affinity in Figure 2. As for *AA*-constraints, large public clouds such as Amazon EC2 and Microsoft Azure allow the SP to choose a coarse-grained geographical location, but not on finer levels such as site or host. To our knowledge, this functionality is currently only also available in [9], [3].

Verma et al. [15] present a power- and migration-cost aware scheduler (pMapper) upon which we have based our contribution. There are a number of differences between their work and ours: (a) our scheduling model may also be applied in a federated rather than an isolated Cloud; (b) the scheduling model presented here has the notion of *AA*-constraints between VMs; and (c) since our model is also usable for federations, it takes other costs than power and migration into account.

Breitgand et al. [2] present a scheduler with support for both affinity and cross-federation capabilities. They have developed Integer Linear Program formulations for placement strategies, and use the COIN-OR solver to obtain solutions. Our approach is different in that it provides a heuristic to determine which VMs should be easiest to migrate, making it suitable for local search algorithms.

Li et al. [16] extend upon the work in [1] by adding support for dynamic rescheduling and using migration to optimize placement of VMs across a multi-provider hosting scenario. Their broker acts on the behalf of a single SP, rather than at the IP level. The impact of using different instance templates (e.g. different VM sizes in Amazon EC2) as Cloud offerings may differ is studied. Since the broker acts on behalf of the SP, it does not have to avoid violating SLAs but instead attempts to minimize service downtime due to cold migrations. Since their model includes the possibility to assign per-VM penalties for migration, the migratability heuristic can be adapted for use within this system.

Existing approaches for monitoring in Clouds are presented in, e.g., [17], [18]. Both present relevant ways of extracting and managing data, but do not employ semantic metadata to achieve cross-Cloud compatibility. Said et al. [19] present a system and algorithm for automatically adding extensible semantic metadata by inferring structure from Globus Grid monitoring data. In addition to architectural differences, the key conceptual difference between that and our approach is that we believe that monitoring system-specific plugins produce richer semantic metadata than a generic algorithm could. Their algorithm infers a structure and annotates the data accordingly, but does not handle input from non-Globus systems and does

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

not aim at making monitoring systems cross-compatible.

Passant et al. [20] use PubSubHubbub to provide close to real-time updates of data sets matching SPARQL queries. The result is turned into Atom feeds, which in turn are published using PubSubHubbub. This approach gives real-time updated streams of specific data, which could be used in conjunction with the MEDICI system to provide access to only relevant subsets of the information.

VII. CONCLUSIONS

This work describes ongoing work on fundamental service management tasks key to federated Cloud environments. We present a hierarchical graph structure representing a service and any placement restrictions placed upon the service components, such as site-level affinity, usable in Cloud federations. This way of structuring a service and defining *AA*-constraints offers a certain amount of control to the SP, which is then enforced by the IP. This facilitates management considerably for an SP compared to multi-provider hosting scenarios.

We define a model for scheduling in Cloud federations that abides by SP-specified *AA*-constraints. We present a heuristic that helps the model determine which VMs are suitable candidates for migration. The model is designed for optimizing placement both within a single site and in a Cloud federation. The heuristic is based on the intuition that the VMs that are most potentially costly in terms of SLA violations are those which are highly active, have *AA*-constraints that require further migrations, and where most data needs to be transferred.

All management of services in Cloud federations, including scheduling, requires cross-site compatible monitoring systems. Current monitoring systems are incompatible in both data format and semantics of what the data represents. To help overcome these issues, we present MEDICI, a monitoring data distribution architecture that annotates data with semantic metadata. Interaction with the data is made simple and flexible e.g. by publishing it to a semantic database upon which SPARQL queries can be made.

ACKNOWLEDGMENTS

The research that led to these results is partially supported by the European Community's Seventh Framework Programme (FP7/2001-2013) under grant agreements no. 215605 (RESERVOIR) and no. 257115 (OPTIMIS) and the Swedish Government's strategic research project eSSENCE. We thank the anonymous referees for their valuable feedback.

REFERENCES

[1] J. Tordsson, R. Montero, R. Vozmediano, and I. Llorente, "Optimized placement of virtual machines across multiple clouds," 2010, submitted for journal publication.

[2] D. Breitgand, A. Marashini, and J. Tordsson, "Policy-driven service placement optimization in federated clouds," IBM Research Report, Tech. Rep. H-0299, 2011.

[3] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan, "OPTIMIS: a holistic approach to cloud service provisioning," in *First IEEE International Conference on Utility and Cloud Computing (UCC 2010)*, December 2010, accepted.

[4] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4:1 –4:11, July 2009. [Online]. Available: http://dx.doi.org/10.1147/JRD.2009.5429058

[5] E. Elmroth and L. Larsson, "Interfaces for Placement, Migration, and Monitoring of Virtual Machines in Federated Clouds," in *Eighth International Conference on Grid and Cooperative Computing (GCC 2009)*. Los Alamitos, CA, USA: IEEE Computer Society, August 2009, pp. 253–260. [Online]. Available: http://dx.doi.org/10.1109/GCC.2009.36

[6] D. Hadas, S. Guenender, and B. Rochwerger, "Virtual Network Services For Federated Cloud Computing," IBM Technical Reports, Tech. Rep. H-0269, Nov. 2009. [Online]. Available: http://domino.watson.ibm.com/library/cyberdig.nsf/papers/3ADF4AD46CBB0E6B852576770056B848

[7] K. Jeffery and B. Neidecker-Lutz, Eds., *The Future Of Cloud Computing, Opportunities for European Cloud Computing Beyond 2010*. European Commission, Information Society and Media, January 2010. [Online]. Available: http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf

[8] I. Brandic, S. Pllana, and S. Benkner, "High-level composition of QoS-aware Grid workflows: an approach that considers location affinity," in *Workshop on Workflows in Support of Large-Scale Science. In conjunction with the 15th IEEE International Symposium on High Performance Distributed Computing, Paris, France*, 2006.

[9] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Vaquero, "Service Specification in Cloud Environments Based on Extensions to Open Standards," in *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, ser. COMSWARE '09. New York, NY, USA: ACM, 2009, pp. 19:1–19:12. [Online]. Available: http://doi.acm.org/10.1145/1621890.1621915

[10] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines," in *VEE '11: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2011)*. ACM, March 2011, accepted for publication.

[11] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, pp. 14–22, 2009.

[12] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," W3C, Tech. Rep., January 2008. [Online]. Available: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

[13] libvirt development team, "libvirt: The virtualization api," December 2005. [Online]. Available: http://libvirt.org/

[14] B. Fitzpatrick, B. Slatkin, and M. Atkins, "PubSubHubbub Core 0.3," February 2010. [Online]. Available: http://pubsubhubbub.googlecode.com/svn/trunk/pubsubhubbub-core-0.3.html

[15] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems," in *Middleware 2008*, ser. Lecture Notes in Computer Science, V. Issarny and R. Schantz, Eds. Springer Berlin / Heidelberg, 2008, vol. 5346, pp. 243–264. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89856-6_13

[16] W. Li, J. Tordsson, and E. Elmroth, "Modelling for dynamic cloud scheduling via migration of virtual machines," 2011, to appear.

[17] S. Clayman, A. Galis, C. Chapman, G. Toffetti, L. Rodero-Merino, L. Vaquero, K. Nagin, and B. Rochwerger, "Monitoring Service Clouds in the Future Internet," in *Towards the Future Internet - Emerging Trends from European Research*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2010, pp. 115–126.

[18] G. Katsaros, G. Kousiouris, S. Gogouvitis, D. Kyriazis, and T. Varvarigou, "A service oriented monitoring framework for soft real-time applications," in *Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on*. IEEE, pp. 1–4.

[19] M. Said and I. Kojima, "S-MDS: Semantic Monitoring and Discovery System for the Grid," *Journal of Grid Computing*, vol. 7, pp. 205–224, 2009, 10.1007/s10723-008-9111-2. [Online]. Available: http://dx.doi.org/10.1007/s10723-008-9111-2

[20] A. Passant and P. Mendes, "sparqlPuSH: Proactive notification of data updates in RDF stores using PubSubHubbub," in *Scripting for the Semantic Web Workshop (SFSW2010) at ESWC2010*, 2010. [Online]. Available: http://www.semanticscripting.org/SFSW2010/papers/sfsw2010_submission_6.pdf

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

IV

Paper IV

Decentralized, Scalable, Grid Fairshare Scheduling (FSGrid)

Per-Olov Östberg, Daniel Henriksson, and Erik Elmroth

Dept. Computing Science and HPC2N, Umeå University
SE-901 87 Umeå, Sweden
{p-o, danielh, elmroth}@cs.umu.se
http://www.cs.umu.se/ds

Abstract: This work addresses Grid fairshare allocation policy enforcement and presents FSGrid, a decentralized system for Grid-wide fairshare job prioritization. The presented system builds on three contributions; a flexible tree-based pol- icy model that allows delegation of policy definition, a job prioritization algorithm based on local enforcement of distributed fairshare policies, and a decentralized architecture for non-intrusive integration with existing scheduling systems. The system supports organization of users in virtual organizations and divides usage policies into local and global policy components that are defined by resource owners and virtual organizations. The architecture realization is presented in detail along with an evaluation of system behavior in an emulated environment. The system is shown to meet scheduling objectives and convergence noise (mechanisms counteracting policy allocation convergence) are characterized and quantified. System mechanisms are shown to be scalable in tests using realistic policy allocations.

Key words: Grid scheduling, Fairshare scheduling, Fair share scheduling, Grid allocation policy enforcement

Decentralized, Scalable, Grid Fairshare Scheduling (FSGrid)

Per-Olov Östberg and Daniel Henriksson and Erik Elmroth

Dept. Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden

Abstract

This work addresses Grid fairshare allocation policy enforcement and presents FSGrid, a decentralized system for Grid-wide fairshare job prioritization. The presented system builds on three contributions; a flexible tree-based policy model that allows delegation of policy definition, a job prioritization algorithm based on local enforcement of distributed fairshare policies, and a decentralized architecture for non-intrusive integration with existing scheduling systems. The system supports organization of users in virtual organizations and divides usage policies into local and global policy components that are defined by resource owners and virtual organizations. The architecture realization is presented in detail along with an evaluation of system behavior in an emulated environment. The system is shown to meet scheduling objectives and convergence noise (mechanisms counteracting policy allocation convergence) are characterized and quantified. System mechanisms are shown to be scalable in tests using realistic policy allocations.

Keywords: Grid scheduling, Fairshare scheduling, Fair share scheduling, Grid allocation policy enforcement

Email address: {p-o, danielh, elmroth}@cs.umu.se [http://www.cs.umu.se/ds] (Per-Olov Östberg and Daniel Henriksson and Erik Elmroth)

1. Introduction

The core idea of fairshare scheduling is to schedule jobs with respect to what fraction of preallocated resource capacity job owners have consumed within a finite time window [12]. Existing schedulers such as Maui [11] and Simple Linux Utility for Resource Management (SLURM) [19] have built-in mechanisms for fairshare, but are not designed to support Grid environments that span multiple administrative domains, utilize heterogeneous schedulers, and require support for site autonomy in allocation policies. This work addresses a need for a global mechanism for Grid allocation policy enactment and presents FSGrid, a system for decentralized fairshare job prioritization that operates on global (Grid-wide) usage data and provides fairshare support to resource site schedulers.

Much work on Grid infrastructure have been directed towards virtualization of job and resource management, but many state of the art Grids still lack adaptability and flexibility in usage policy enactment. Rigidity in allocation mechanisms can effectively restrict many of the main use cases for Grids and, e.g., force end-users to perform manual resource selection to meet usage allocation criteria not supported by automated brokers. To facilitate the Grid vision of transparency in end-user resource utilization, policy enactment mechanisms that virtualize Grid-level usage allocation are required. To facilitate scalability in system deployment and administration, Grid fairshare enactment systems should also allow delegation of policy administration, use scalable fairshare calculation algorithms, and support management of Grid-scale volumes of usage data.

The proposed system provides a flexible capacity allocation policy model that maps organizational structures directly to policy specifications. The policy model separates policy specifications into local and global components, and delegates policy component administration to policy actors, e.g., Virtual Organizations (VOs) [10] and projects. Resource sites mount global policy components onto local policies to form policy trees, which allows global policy allocation updates (performed by policy actors) to be transparently propagated to resource sites.

The fairshare algorithm operates on usage data and compares consumed resource capacity to policy-defined capacity allocations. Fairshare is calculated for each level in the tree-based policy model and enforced top-down, ensuring fairshare balance between policy subgroups to take precedence over balance within subgroups. As local policy components are defined by site administrators and form top levels of policy trees, site owners retain full control over site resources.

FSGrid employs an architecture for distributed storage of usage data, and maintains periodic summaries at resource sites. With minimal demands on the content of usage data, the system integrates seamlessly with accounting systems, facilitating automated tracking of Grid-level usage data. FSGrid places a component close to resource site schedulers that local scheduling mechanisms invoke to replace existing fairshare calculations, imposing minimal changes to existing deployment environments. The main building blocks of FSGrid are:

- A Grid usage policy allocation model that supports recursive delegation of policy administration.
- An algorithm for efficient calculation of job prioritization from usage data and allocation policies.
- A decentralized and distributed architecture for dynamic fairshare policy enactment that implements the proposed fairshare algorithm and integrates with minimum intrusion into existing high-performance computing resource environments.

The evaluation presented in this work assumes a general model of Grid environments built on High-Performance Computing (HPC) resource sites, where jobs are fed from a batch system into a (cluster) scheduler. While this model is representative for many current Grid environments, the proposed system is not limited to HPC deployment Grids. The proposed system can be utilized by any system that performs execution order prioritization of jobs. The proposed system contains no functionality for advanced scheduling mechanisms, e.g., job preemption, and is to be viewed as an independent job prioritization component rather than a full policy enforcement or job scheduling mechanism.

Ordering of jobs with respect to differences between usage allocation and resource consumption allows schedulers to achieve a fairshare job prioritization semantic of "least favored first". This creates a global self-adjusting policy enactment mechanism that helps users receive resource capacity as defined by policy allocations. By definition of an allocation policy model for VOs, a fairshare algorithm operating on the policy model, and an architecture for distribution and decentralization of policy enforcement, we extend an existing fairshare mechanism to Grid level.

In FSGrid, we define fairness in terms of convergence of resource consumption to policy-defined preallocations over time. As a point of departure, this work builds on earlier efforts [9] where preliminary versions of the

policy model and algorithm are presented. A comprehensive differentiation and discussion of new and prior results is given in Section 6.2.

The rest of the paper is structured as follows. In the first sections we present the building blocks of the FSGrid system; a tree-based policy model (Section 2), an algorithm for efficient calculation of fairshare vectors (Section 3), and a decentralized architecture for scheduler-based Grid allocation policy enactment (Section 4). These are followed by a performance evaluation and a discussion of the proposed system in sections 5 and 6, and a survey of related work in Section 7. Finally, Section 8 outlines possible directions for future work, and the paper is concluded in Section 9.

2. A Tree-Based Usage Policy Model

Grids are typically formed through joint collaborations of autonomous resource sites. The amount of resources contributed to a specific collaboration normally differs between sites, and may vary over time. Grid policy models, i.e. mechanisms for mapping user identities to resource allocations, must allow site administrators to specify resource allocations on multiple levels, e.g., between local and Grid jobs, or different Grid collaborations (e.g., VOs). As Grid user bases are usually formed as VOs, Grid policy mechanisms are required to adapt to dynamic changes in VO structure.

As illustrated in Figure 1, FSGrid employs a model for specification of usage allocations in *policy trees*. Policy tree nodes contain tuples of *VO identity strings*, i.e. strings uniquely identifying a VO entity (e.g., a user or a project), and *usage share values*. A usage share value expresses a relative usage preallocation of resource capacity within a *policy group* (a set of VO identities that are policy tree siblings). The user $U4$ allocation of 0.2 in Figure 1 is interpreted as $U4$ being allocated 20 percent of whatever resource capacity (e.g., monthly CPU hours) is allocated to project $P1$.

This model allows VOs to map internal structure directly onto policy trees, and express both organizational hierarchy (tree structure) and relationships between and within policy groups (node share values) in a single structure. There are no limitations on policy organization other than VO identities being unique within tree levels, i.e. within policy groups or projects.

Expression of allocation quotas in relative usage metrics (e.g., share percentages) rather than absolute capacity metrics (e.g., CPU hours) virtualizes both the currency used in the system and allocation of resource site capacity. Separation of allocation quotas from resource capacity metrics allows policy quota allocations to be mapped to custom metrics, provides a semantic for reallocation of unused policy allocations, and insulates allocation enforcement mechanisms from volatility in resource site capacity.

As FSGrid policy trees express relative share ratios and make no assumptions of tree structure, policy trees can be constructed from multiple sources by mounting subtrees onto leaf nodes in a policy tree (see Figure 1). FSGrid makes a semantic distinction between *local* and *global share policies*. Local share policies are root policies defined by resource site administrators for individual resource clusters. Global share policies are independent policy trees defined by VOs, and are mounted onto local policy trees by resource site administrators (also illustrated in Figure 1). Local policies express what global policies to enact and relative resource allocations between them. Local share policies may have local queue components that allow site administrators to reserve resource capacity for local (non-Grid) jobs. Global policy trees express structure and allocations for VO components, e.g., groups, projects, and users.

As policy tree construction can be distributed and performed recursively, FSGrid delegates policy component (subtree) administration to policy actors. Delegation of policy specification allows policy actors, e.g., individual projects in a VO, to define policy components (subtrees) and mount these onto (leaf) nodes in parent policy trees, i.e. updating usage policy allocations without involving resource site administrators. Mounting global policy components to local policy trees allows resource site owners to subdivide and allocate resource site capacity shares to virtual organizations, which can further subdivide and allocate resource site capacity shares recursively within their organization.

Mounting of policy components onto policy trees does not violate the node peer uniqueness criteria of the policy model as subtree root nodes are overwritten by policy tree nodes in the mounting process. Paths in policy trees uniquely qualify both VO identities (the bottom path node) and chains of relationships between VO identities and policy ancestors.

3. A Grid Fairshare Algorithm

Fairshare scheduling relies on prioritization of jobs with respect to consumption of resource capacity preallocations. In FSGrid, job prioritization is performed through comparison of *fairshare vectors*, vectors of fairshare balance values calculated from paths in *fairshare trees*. Fairshare trees inherit structure from policy trees and are calculated from comparisons of policy trees and

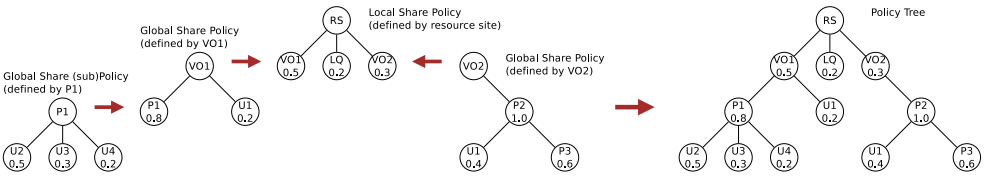


Figure 1: Delegation of policy specification to policy actors. Resource capacity allocations are subdivided recursively in usage shares. Resource site *RS* local share policy contains preallocated usage shares for virtual organizations ($VO1$ and $VO2$) and local job queue (LQ). Administration of policy components is delegated to organization and project administrators.

historical usage data. As paths in policy trees define ancestries of VO identities, comparison of fairshare vectors offer a computationally efficient way to simultaneously perform scheduling prioritization on multiple levels in policy trees.

The FSGrid fairshare algorithm performs calculation of fairshare vectors in two steps. First, a fairshare tree is calculated (once per resource site, illustrated in Figure 3) from an FSGrid policy tree and historical usage data. Second, fairshare vectors representing each VO identity in the system are calculated from the fairshare tree (once per VO identity, illustrated in Figure 4), and associated to jobs.

3.1. Fairshare Tree Calculation

Calculation of fairshare trees is done in two steps. First, a *usage tree* is constructed by recursively (bottom-up) replacing all node values in a policy tree with a cumulative usage sum. This value is calculated as the sum of all usage data found in the usage time window for the node VO identity and the sum of all child node values. To facilitate comparison of usage and policy data, node values are normalized to $[0, 1]$. Normalization is performed by replacing each node value with the node's relative share of the sum of all node values on the tree level. If no usage data is found (i.e. all sibling nodes have value zero), all sibling nodes in the tree level receive equal shares. Like in policy trees, all tree level node values sum to 1 after normalization.

Second, a fairshare tree is calculated by node-wise application of a *fairshare operator* on the policy and usage trees (illustrated in Figure 3). Fairshare operators compare share values from policy and usage trees and quantify a distance from the current to the ideal system fairshare balance state (where all users utilize resource capacity according to policy capacity preallocations). The policy and usage trees are identical in structure, and have node values in $[0, 1]$. Node values in the resulting fairshare tree are in $[-1, 1]$, and quantify a difference between policy usage preallocation and actual resource consumption (as defined by the fairshare operator used). Node value sign indicates direction (positive values underuse, negative overuse), and magnitude quantifies distance to policy-usage balance. All tree level node values in fairshare trees sum to 0. Like a policy tree contains all information required for policy enactment for a VO or a resource site, a fairshare tree contains all information required to perform fairshare prioritization of jobs on a resource site.

3.2. Fairshare Vector Calculation

Once a fairshare tree has been calculated, individual VO identity fairshare vectors are calculated (illustrated in Figure 4). As paths in fairshare trees uniquely define ancestries of VO identities, combining fairshare tree node values (top-down) along a tree path creates vectors that contain fairshare information for hierarchies of VO identities. After vector calculation, node values (x) are transformed to integer elements (y) as

$$y = floor((\frac{x+1}{2}) * 9999) \quad (1)$$

where

$$x \in [-1, 1]$$

$$y \in [0, 9999]$$

This results in integer vectors that can be serialized to strings and compared lexicographically. The value 9999 is an upper limit constant determining the numerical resolution of vector element integer representations.

For arithmetic comparison of vectors, where vectors are projected to one-dimensional value spaces, vectors are required to be of uniform length. Therefore, vectors are appended zero value elements until they reach maximum vector length (defined by fairshare tree depth). Zero is chosen as pad value as it expresses policy-usage

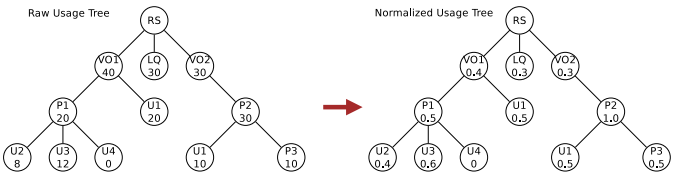


Figure 2: Construction of usage trees from (distributed) usage data is done in two steps: 1. Raw usage trees inherit structure from policy trees and node values are defined by cumulative summation of usage data for all usage identities at or below the current node. In the illustration, project *P2* consumes 10 usage credits. 2. Usage trees are normalized to enable policy comparison through recalculation of node values as relative shares of node tree level usage data.

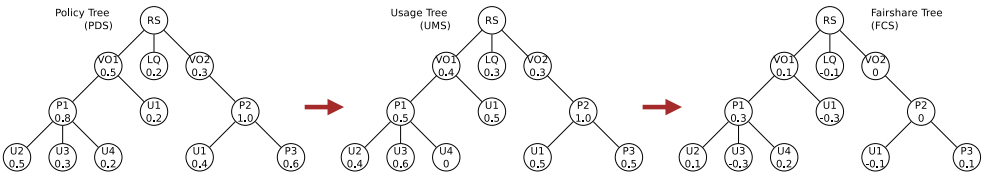


Figure 3: The FSGrid fairshare calculation algorithm. Fairshare trees are calculated by node-wise application of a fairshare distance measure operator on the policy tree and the usage tree, in the illustration the absolute fairshare operator $p - u$. Done once per Grid site and scheduling step.

balance in fairshare trees. As illustrated in Figure 4, padding is performed prior to transformation to integer vectors in the vector extraction algorithm.

Prioritization of jobs based on fairshare vector comparison results in hierarchical ranking of VO identities. Vector comparisons express differences in policy-defined preallocations and actual resource capacity consumption on multiple policy levels. As comparison is done on job ownership VO identity level, all jobs owned by the same VO identity receive the same priority.

In this framework, fairshare scheduling can be viewed as an optimization problem where the distance from each VO identity's usage state and the system balance axis are sought to be minimized simultaneously. By prioritizing jobs by fairshare distances, a scheduling policy of "least favored first" is enacted. The term *convergence* is in this context defined to refer to VO identities' resource capacity consumptions approaching policy-defined usage preallocations over time. Conversely, any mechanism counteracting system convergence in this context is defined to be *convergence noise*.

3.3. Fairshare Distance Measure Operators

For fairshare job prioritization, a mechanism to quantify differences between usage share preallocations and resource usage is required. To construct a metric for comparison, FSGrid defines a two-dimensional value space spanned by unit basis vectors for policy share preallocations (p) and resource capacity consumption (u). The system balance state, where resource consumptions equal policy allocations, forms an axis ($u = p$) transecting the value space diametrically.

By ordering VO identities by the distance from their current usage state (a function of p and u) to the system balance axis ($u = p$), a fairshare job prioritization order is established. For distance measurement, FSGrid defines a fairshare operator (d) constituted by an absolute (d_a) and a relative (d_r) component. To increase system configurability, relative operator component influences are regulated by a weight (k).

$$d = kd_a + (1 - k)d_r \tag{2}$$

where

$$d_a = p - u \tag{3}$$

$$d_r = \begin{cases} \left(\dfrac{p-u}{p}\right)^2 & \text{for} \quad u < p \\ 0 & \text{for} \quad u = p \\ -\left(\dfrac{p-u}{u}\right)^2 & \text{for} \quad u > p \end{cases} \tag{4}$$

$$k, p, u \in [0, 1]$$

$$d, d_a, d_r \in [-1, 1]$$

While any arbitrary operator (with arbitrary value

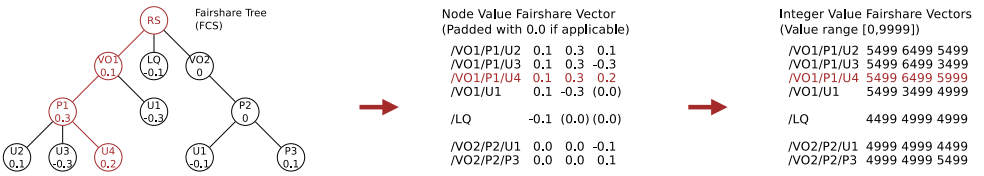


Figure 4: VO identity fairshare vectors are calculated and padded to uniform length. Node values ($[-1, 1]$) are converted to integer values ($[0, 9999]$). Fairshare vectors are calculated once per VO identity in fairshare trees.

space) may be chosen for fairshare distance measurement, operator selection impacts complexity and design of the system. For example, uniform and symmetric value spaces make distance interpretation intuitive, zero distance balance points facilitates padding of fairshare vectors, and unit distance magnitude facilitates scaling of fairshare balance values. Conceptually the absolute fairshare operator can be seen as a geometrical measurement of the distance between resource consumption and policy allocations in usage credits. The relative fairshare operator expresses a ratio between resource capacity consumption and policy preallocations.

The requirement for a combined operator stems from the behavior of the individual operator components. In situations where a VO identity does not utilize allocated capacity, the absolute operator degenerates and divides unused allocations evenly among VO identity peers. In situations where no usage data is available (e.g., at startup) the absolute operator favors users with large usage shares. In situations where zero policy allocations are assigned VO identities with reported usage, the relative operator yields a maximum distance regardless of differences in usage consumptions. By design, the relative operator has a higher resolution far from balance, and a lower resolution near balance. Combining the two operator components allows FSGrid to operate more robustly, and provides administrators the ability to customize the fairshare operator.

3.4. Combining Job Prioritization Mechanisms

As defined here, fairshare scheduling implies only a prioritization order for jobs. Jobs with low fairshare values may be scheduled if there are resources available and no jobs with higher fairshare prioritization value in queue. Jobs are by this mechanism not preempted or stalled, and fairshare scheduling is to be considered a soft scheduling mechanism. If policy fairness is more important than resource utilization, schedulers may combine fairshare prioritization with external mechanisms that, e.g., reject jobs with fairshare values below a certain threshold.

Some schedulers, such as Maui and SLURM, calculate a linear combination of multiple scheduling factors to determine job prioritization order. In these cases, a scalar fairshare rank value computed by the FSGrid algorithm can be used as a fairshare component in the linear combination. If so, the fairshare vector must be projected onto a limited value range to restrict the final prioritization value's range, which may affect the numerical stability of fairshare prioritization. To avoid this, projection of the fairshare balance values (fairshare vector elements) to a more restricted value range may be replaced with an algorithm that assigns values to vector elements according to group-wise sort order. This will project the fairshare vector to a truncated value range, preserving vector sort order while truncating distances between vectors uniformly. Typically, schedulers that use linear combinations of scheduling factors allow site administrators to configure weights to determine to what extent fairshare factors influence job prioritization.

4. A Decentralized Grid Fairshare Architecture

The policy model and algorithm of sections 2 and 3 provide a mechanism for fairshare prioritization of jobs based on usage allocation and resource consumption. As usage allocation policies are constructed from distributed policy components, and the algorithm operates on usage data from multiple distributed resource sites, an architecture managing distribution of data and computations is required.

As illustrated in Figure 5, the architecture of FSGrid is designed as a distributable Service-Oriented Architecture (SOA) where blocks of functionality in the FSGrid fairshare algorithm are identified and exposed as services. The FSGrid architecture contains three major blocks of functionality; policy administration, usage data monitoring, and fairshare vector calculation; which also constitute integration points between FSGrid and the deployment environment.

To facilitate computational efficiency and reduce communication overhead of the system, a number of ob-

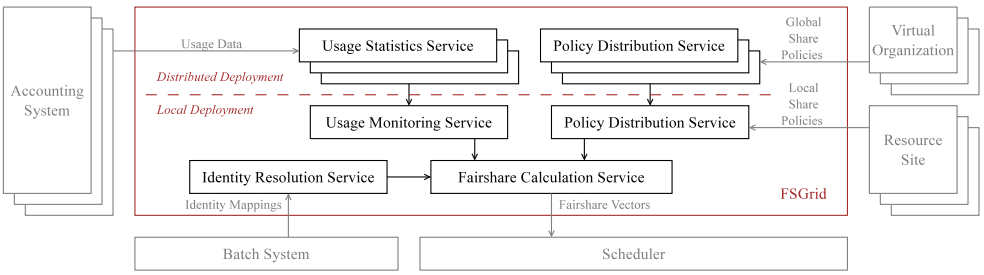


Figure 5: The FSGrid architecture. System functionality is segmented into distributable services. The system integrates with cluster schedulers, requires policy definitions from organizations, usage data from Grid accounting systems, and (optionally) identity mappings from batch systems. Deployment patterns are expected to be site-dependent.

servations about the interaction patterns of the functionality blocks can be made. Fairshare vectors are required for job prioritization and should be recalculated whenever updated policy allocations or usage data are available. As schedulers require access to fairshare vectors whenever scheduling decisions are made, e.g., when job queues change or periodic scheduling cycle events occur, the fairshare vector calculation block should be located close to the scheduler. The policy administration and usage data monitoring blocks are by nature distributed, but should for reduction of communication overhead have a cache component close to the fairshare vector calculation block.

The computational complexities of computing fairshare trees and vectors are low, and both operations can be precomputed and results cached, making them well suited for implementation in Web Services. Calculation of the fairshare tree is performed once per resource site and scheduling step, and calculation of fairshare vectors is performed once per VO identity owning a job in the scheduling queue.

Note that the design of the system does not assume coordination of component actions, or synchronization of distributed state, but rather realizes a set of autonomous components that combined form a decentralized fairshare architecture. Global fairshare resource allocation is enacted through concurrent, asynchronous local computations on distributed data.

To minimize the system deployment footprint, all services are designed to integrate non-intrusively with existing infrastructure and minimize network traffic required by the system. Service deployment patterns are expected to vary from site to site, but are recommended to be based on the pattern illustrated in Figure 5 to minimize communication overhead.

4.1. Architecture Components

As illustrated in Figure 5, FSGrid is constituted by five services and a set of plug-ins for scheduler prioritization, usage data submission, and identity resolution. To facilitate seamless integration into existing HPC deployments, the architecture is implemented in Java and exposes service functionality through WSDL SOAP Web Services deployed in Apache Axis2 service containers. Integration with HPC cluster schedulers (currently Maui and SLURM) is done through injection of FSGrid clients into scheduler exposed prioritization customization points.

4.1.1. Policy Distribution Service (PDS)

The Policy Distribution Service (PDS) provides a service interface to FSGrid usage policy allocations. Internally, the PDS collates policy components from multiple sources, e.g., XML files, HTTP web resources, other PDSs; assembles a policy tree; and publishes policies through the service interface. As multiple PDS may be chained, and data read remotely, the PDS provides a flexible mechanism for delegating policy definition to VO and site administrators. To FSGrid and FSGrid clients, the PDS provides an easy to use interface for policy retrieval, and can be to, e.g., monitor updates in policy allocations.

4.1.2. Usage Statistics Service (USS)

The Usage Statistics Service (USS) is designed to provide time-resolved histograms of usage data on a per-user basis. To reduce the amount of data, the service interface accepts updates in a format semantically equivalent to summaries of Open Grid Forum (OGF) Usage Records [17], and exposes usage summaries for requested time windows. Internally, the USS stores

usage histograms for known users in a database, and maintains a usage summary cache to minimize invocation response time. The USS is the only required part of FSGrid that receives input data from the surrounding system environment. As usage data constitutes the currency that drives FSGrid fairshare, it is vital to FSGrid system coherency that each job usage record is only reported to a single USS. As the USS provides a histogram-based view of historical usage data, it can be used by FSGrid services and clients to assess usage statistics for individual VO identities on individual resource sites.

4.1.3. Usage Monitoring Service (UMS)

The main task of the Usage Monitoring Service (UMS) is to provide a service interface for computation of (normalized) usage trees from policy trees. Internally the UMS compiles data from a set of known USSs, maintains a database of USS usage summaries, a time-resolved per-user usage cache, a cache of previously known policy trees, and agents to monitor USSs and precompute usage trees. The UMS also maintains a customization point for moderation of usage data influence through a time window and usage decay function plug-in. The UMS provides an interface for summarizing usage records from multiple (USS) data sources and mapping these to (provided) usage policies, and can be used by FSGrid services and clients to get normalized usage data views.

4.1.4. Identity Resolution Service (IRS)

Key to enabling fairshare scheduling of jobs in FSGrid is to be able to access historical usage records for VO identities. As VO identities may be translated to local cluster or site users when jobs are dispatched to batch queues, schedulers may lack access to VO identities. The IRS exposes an interface for storing and accessing VO identity to job associations, and is primarily used to resolve job ownerships. Use of the IRS in FSGrid is optional. If a scheduler has access to VO identity job ownership data, these may be used directly when requesting scheduling prioritization information.

4.1.5. Fairshare Calculation Service (FCS)

The Fairshare Calculation Service (FCS) offers a flexible service interface that provides access to the FSGrid policy-based fairshare tree, fairshare vectors for specified VO identities (or jobs), and preformatted fairshare tuples that contain VO identities, fairshare vectors, and scalar fairshare prioritization values. The rich interface of the FCS is designed to facilitate flexibility in implementation of scheduler integration plug-ins. Internally, the FCS maintains caches for job identifier to VO identity maps, policy, usage, and fairshare trees, as well as agents for monitoring services (PDSs and UMSs) and precomputing fairshare trees. The FCS allows configuration of UMS and PDS connections, PDS deployments, and monitoring scheduling intervals.

4.1.6. Integration Plug-Ins

In addition to the services of FSGrid, a set of integration plug-ins is also considered part of the FSGrid architecture. Depending on the FSGrid deployment environment, integration plug-ins for scheduler job prioritization, usage data submission, and VO identity resolution may be required. Design of scheduler plug-ins depend on scheduler architecture, but typically consist of an FCS client implemented in the same language as the scheduler and possibly routines for calculation, transformation, and caching of fairshare prioritization data. Design of plug-ins for usage data submission and VO identity resolution depend on accounting system and scheduler architecture, and will typically consist of USS and IRS clients.

As many Grid computing environments build on existing HPC deployments, which typically are required to maintain HPC interfaces (e.g., batch systems) in coexistence with Grid interfaces, it is vital to design Grid systems to impose a minimum intrusion level when integrating Grid components with existing HPC deployments. The FSGrid architecture is designed to have as few and simple integration points as possible while still maintaining compatibility with a general model for HPC deployment based Grid environments.

Typical Grid FSGrid integrations include

- Replacement of a local scheduler (fairshare) job prioritization mechanism with an FCS invocation client.
- Injection of a mechanism for submission of usage data to the USS. This can be done in multiple ways, e.g., through a scheduler job monitoring plug-in, or a resource site or Grid accounting system.
- Optional injection of a job ownership resolution component. If VO identity job ownership data is not available to the scheduler, a job ownership mapping between job and original VO identity can be stored in the IRS. This data can be submitted at any point prior to invocation of the FCS. Submission is typically expected to be done by the system responsible for translation of VO identities to local resource site users, e.g., a batch system.

4.2. (Concurrency in) Data and Control Flow

Data and control flows of an FSGrid deployment consist of five autonomous and concurrent processes:

1. A set of PDSs monitors a set of data sources and periodically compiles policy trees.
2. A set of USSs receives (summarized) usage reports for jobs and builds time-resolved usage histograms.
3. A UMS monitors a set of (local or remote) USSs, periodically retrieves updates, and assembles usage summaries. The UMS precomputes usage trees for known policy trees, and on demand for unknown policy trees (which are added to the cache structure).
4. An IRS receives VO identity job ownership data and maintains a directory for ownership resolution.
5. An FCS monitors a PDS and periodically retrieves policy trees and calculates usage (via a UMS) and fairshare trees. The FCS maintains a cache of precomputed fairshare vectors (based on precomputed fairshare trees), and does not compute fairshare vectors for unknown VO identities.

The data required to drive the system, usage data and usage policy allocations, are provided by accounting systems and VO, project, and resource site administrators respectively. FSGrid assumes that jobs are scheduled in an order influenced by fairshare prioritization and that usage costs for all jobs are reported to the system. Should resource sites utilize FSGrid to prioritize jobs without reporting usage data, resource consumption costs for jobs running on such sites do not contribute to fairshare calculation results and imbalances in global resource consumption may occur. Conversely, should resource sites report usage data without utilizing FSGrid as a job prioritization mechanism, global fairshare convergence will suffer oscillations correlated in size to resource site capacity. As the core balancing mechanism of FSGrid is self-adjusting, global fairshare balance will converge over time.

Specification of usage policies can be seen to be a largely manual process, while usage data submissions are expected to be fully automated. Through these five processes, the FSGrid system provides an automated, decentralized, and self-adjusting mechanism for Grid-wide fairshare enactment of usage policy allocations.

4.3. Time Window and Decay Function

As illustrated in Figure 6, FSGrid defines a finite usage data time window (typically a configurable amount of days into the past) to restrict the influence of historical usage data on the fairshare mechanism. As also illustrated, FSGrid employs a customizable usage decay function to modulate how usage statistics influence the fairshare mechanism. The time window width limits the scope of usage statistics influence (data outside the time window does not affect FSGrid behavior). The granularity of the time window histogram slots affect the resolution of the fairshare mechanism. The usage decay function modulates usage statistics by, e.g., increasing or decreasing influence of more recent usage statistics on system behavior. In the FSGrid architecture, both time window parameters are configurable, and the usage decay function is exposed as a customization point in the UMS. Further study of the impact of usage decay functions in this context is subject for future work.

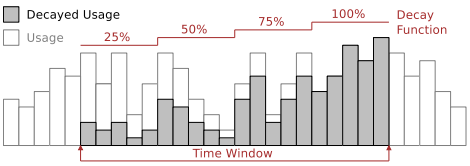


Figure 6: Usage data histogram time window. Usage decay functions modulate influence of usage data.

5. Evaluation

To evaluate core system functionality and isolate noise sources (i.e. mechanisms counteracting system convergence), a number of tests designed to quantify aspects of FSGrid's technical performance are employed. These tests are run in an emulated system environment and are designed to introduce and illustrate system mechanics. As the purpose of this evaluation is to evaluate system ability to enact policy allocations in a distributed environment rather than demonstrate system integration in a production deployment, use of an emulated system environment is sufficient. In the evaluation, the follow tests are performed:

- Noise characterization tests (Section 5.1). Investigate and characterize FSGrid noise mechanisms.
- Noise interaction tests (Section 5.2). Investigate interaction between different noise types and illustrate impact of system deployment patterns on system performance.
- Policy enactment tests (Section 5.3). Investigate FSGrid ability to enact policy allocations in decen-

tralized multi-site deployments employing multiple asynchronized concurrent schedulers. Quantify and evaluate FSGrid ability to adapt to dynamic changes in policy allocations and distributed system failures.

- Scalability tests (Section 5.4). Investigate FSGrid ability to cope with realistically sized policy allocations and quantify system scalability in the presence of large amounts of usage data and updates.

All evaluation tests are performed on a set of four identical 1.8 GHz quad core AMD Opteron CPU, 4 GB RAM machines, interconnected using a Gigabit Ethernet network. For functionality tests, an additional set of four identical 2 GHz AMD Opteron CPU, 2 GB RAM machines, interconnected with a 100 Mbps Ethernet network are used. All machines are running Ubuntu Linux and Axis2 1.5. The Java version used in tests is 1.6, and Java memory allocation pools range from 512 MB to 1 GB in size. For system integration tests SLURM 2.1.2 is employed as batch system and cluster scheduler.

Functionality tests are performed using a discrete-event simulator emulating an execution environment consisting of a batch system, a cluster scheduler, a cluster, and an accounting system. The batch system registers (in the IRS) and feeds the scheduler sets of jobs. The scheduler invokes the FCS to prioritize jobs and allocates them to cluster hosts. The accounting system submits usage reports to the USS upon job completions. Job start and end timestamps are used to evaluate FSGrid ability to enact resource capacity allocations.

Simulation job arrival models saturate scheduling queues in the sense that schedulers have access to at least one job for each usage policy VO identity at all times. Single-site system emulations are run on a single host, multi-site system emulations as a set of non-communicating systems run concurrently on multiple hosts. Cross-site synchronization is performed exclusively by UMSs, which have access to USSs for all sites, emulating a distributed Grid configuration.

All tests are, unless stated otherwise, run using identical parameter sets and the policy tree illustrated in Figure 1. USS and UMS update intervals are set to 1 second, usage time windows are 10 slots wide and set to a granularity of 1 hour (system wall clock time), all clusters have a single host, and job lengths are either fixed to 1 or stochastic and uniformly distributed between 1 and 5000 time units long. To eliminate them as parameters in measurements, absolute and relative fairshare operators are equally weighted ($k = 0.5$). Usage decay is disabled (i.e. usage decay function is constant $y = 1$), and the usage cost metric used is job length (CPU time).

Job failures do not affect FSGrid convergence rates as failed jobs do not get reported to the accounting system and appear as diminished resource capacity.

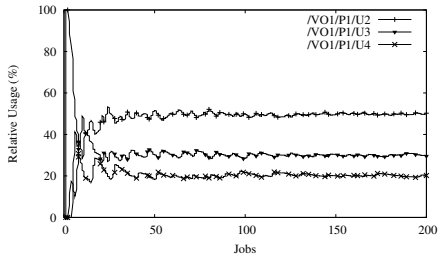
5.1. Noise Characterization

As we refer to system ability to over time enact policy-defined resource capacity allocations as system convergence (to policies), we define any mechanism counteracting this process as convergence noise. To illustrate system convergence to policy allocations, we isolate policy (sub)groups, i.e. groups of nodes with a common parent, and render cumulative resource consumption for individual VO identities as a function of number of jobs run in the group. To maximize the influence of noise in measurements, we isolate the policy subgroup containing the VO identity with the lowest total usage share ($P1$ in Figure 1).

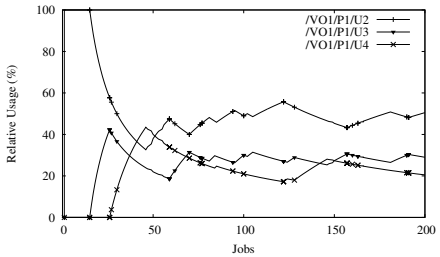
In FSGrid, there are two primary mechanisms counteracting system convergence, *variance in job usage costs* and *usage data update latencies*. To isolate impact of variance in job usage cost, we emulate a single-site FSGrid deployment with stochastic job usage costs drawn from a uniform [1,5000] probability distribution. To eliminate impact of usage data update latencies on system convergence, each scheduling step is delayed to allow usage data updates from prior jobs to propagate to the FCS between scheduling steps. As illustrated in Figure 7a, usage cost variance amplifies oscillations in system convergence. When compared to ideal convergence, differences in usage costs manifest as additive noise in convergence adjustments. In Figure 7a, job usage cost variance noise is illustrated as vertical offsets in convergence oscillations.

To isolate impact of usage data update latencies, we emulate a single-site deployment with uniform job usage cost (cost = 1) and UMS and FCS update delays designed to allow approximately 10 jobs to be scheduled between FCS usage data updates. As the FSGrid job prioritization mechanism operates on usage data for completed jobs, i.e. has no memory for recent scheduling decisions or prediction mechanism for costs of running jobs, usage data update latencies result in multiple subsequent scheduling decisions being taken on the same usage data. As illustrated in Figure 7b, this results in amplifying convergence oscillations and significantly lowering system convergence rate. When compared to ideal convergence, usage data update latencies manifest as multiplicative noise in convergence adjustments and a divisible reduction in convergence rate.

Prior work [9] suggests that including cost for scheduled and running jobs in prioritization calculations reduce impact of usage data update latency noise on sys-



(a) Job usage cost variance noise. Variations in job usage costs cause convergence adjustments to overshoot and amplify convergence oscillations.



(b) Usage data update latency noise. Update latencies reduce granularity of convergence adjustments and delay system convergence.

Figure 7: Noise characterization. Cumulative resource consumption as function of scheduled and run jobs. Convergence to usage policy allocations for VO identities designated in legend. Illustration capped to region of interest.

tem performance. Future work includes evaluation of different strategies for inclusion of this approach in multi-site FSGrid deployments.

5.2. Noise Interaction

Under realistic FSGrid operational settings both job usage cost variance and usage data update latencies are likely to be present. To investigate noise interaction, we emulate a single-site FSGrid deployment with stochastic job lengths and usage update latencies. As illustrated in Figure 8a, usage data update latencies add a multiplicative component to usage cost variance noise. Impact of noise is amplified by lowered convergence rate.

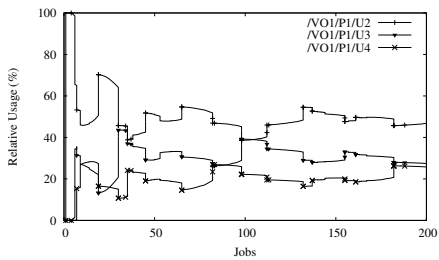
To evaluate noise interaction in decentralized Grid environments, we emulate a four-site FSGrid deployment with stochastic job lengths and usage data update latencies. As illustrated in Figure 8b, parallelism of concurrent scheduling amplifies update latency noise, in this experiment delaying system convergence by a factor of 10. As the number of jobs scheduled between usage data updates determine impact of update latency noise, large numbers of computational resources per scheduler skew system convergence at startup. As job lengths constitute lower bounds for usage data update latencies, excessive job lengths amplify update latency noise. For multi-site settings, concurrent scheduling with synchronized update schedules amplify update latency noise. Conversely, asynchronicity in multi-site update schedules allow parallel processing of usage updates to increase update frequencies and mediate impact of usage data update latency noise.

These experiments are run in an artificial environment, but outline a few interactions between mechanisms in the FSGrid fairshare job prioritization system. Scheduling jobs after the principle of "least favored first" creates a self-adjusting system that over time distributes resource capacity after policy allocations. Noise from job usage cost variance and update latencies lower system rate of convergence by affecting the convergence adjustments (i.e. order in which jobs are run). Number of hosts, sites, job lengths, as well as frequency and synchronicity of usage data update schedules may serve to amplify convergence noise. Over time, relative impact of each noise source and type lessen, as more usage data affect scheduling prioritization. As long as usage data time windows are large enough to contain enough data for the system to converge, the system remains stable.

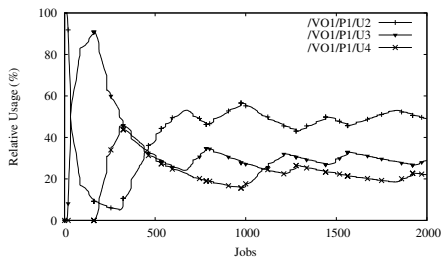
5.3. Policy Enactment

To evaluate system ability to respond to external events such as dynamic changes in site availability or policy allocations, we emulate an eight-site FSGrid deployment with stochastic job lengths and usage data update latencies over an extended period of time. To study impact of site volatility, four sites are removed after approximately 25000 jobs are scheduled. After approximately 50000 jobs are scheduled, the local allocation policy *RS* is altered to transfer 10 percent from each of the allocations for $VO1$ and LQ to $VO2$.

As illustrated in Figure 9a, eight concurrent schedulers cause significant initial convergence noise. At approximately 25000 jobs four schedulers are removed, and convergence noise is reduced (also visible at approximately 10000 jobs in Figure 9b). At approximately 50000 jobs the usage policies are updated and all schedulers adapt to new scheduling priorities. It takes approximately the same amount of jobs currently in the time window to reach the level of convergence achieved

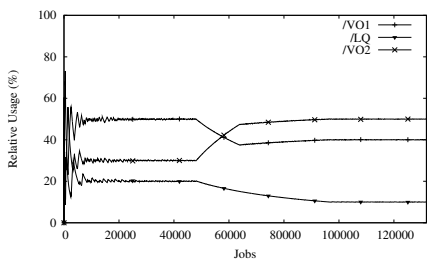


(a) Interaction of usage cost variance and update latency noise. Impact of usage cost variance noise amplified by delayed convergence.

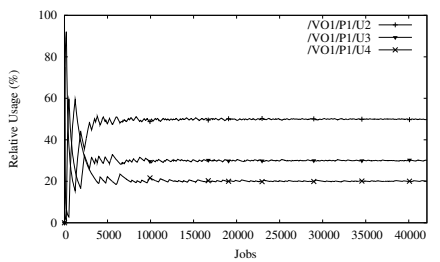


(b) Concurrent scheduler noise augmentation. Impact of usage data update latency noise amplified by parallelism in scheduling.

Figure 8: Noise interaction. Cumulative resource consumption as function of scheduled and run jobs. Convergence to usage policy allocations for VO identities designated in legend. Illustration capped to region of interest.



(a) Adaptation to site failures and updated usage policy allocations.



(b) Adaptation to site failures and subgroup isolation.

Figure 9: Policy enactment. Cumulative resource consumption as function of scheduled and run jobs. Convergence to usage policy allocations for VO identities designated in legend. Illustration capped to region of interest.

before the policy shift. As also illustrated in Figure 9a, convergence rate is a function of the relative share ratio, the VO identity with the lowest policy allocation (LQ) converges slowest.

As illustrated in Figure 9b, which illustrates policy group $P1$ of Figure 9a, altering the policy allocation of an individual policy group does not affect other groups in the same policy tree. Note that this simulation contains multiple shifts of the usage data time window, which do not visibly affect system convergence.

5.4. Scalability Tests

To evaluate FSGrid ability to function in production environments, we run large scale tests over longer periods of time using realistically sized policy allocations and system configurations. System convergence is validated for tests using policy allocations with thousands of users running millions of jobs.

For tests using a balanced policy tree (symmetrically distributed users with equal allocation shares) containing 1000 users, 100 projects, and 10 VOs, the system is shown to converge and exhibit stable performance consistent with the behavior illustrated in tests using smaller policy trees. System behavior is shown to be deterministic and stable in tests using more than 4 million jobs. First 5000 jobs of such a test are shown in Figure 10. Tests using symmetric policy trees with 1000 users and equal allocation shares show faster convergence rates than tests using small asymmetrical policy trees. Exact system convergence rate depends on a number of factors including, e.g., policy tree shape, usage allocation share variance, job length variance, update delays, and site synchronicity, and is considered out of scope for this work. Further scalability and integration tests in production environments, as well as analysis of convergence factors and formulation of convergence rate formulas are subject for future work.

6. Discussion

The system evaluation of Section 5 demonstrates technical aspects of FSGrid and shows how Grid-wide fairshare job prioritization can be realized. While this evaluation is performed in an emulated environment, the evaluation demonstrates key aspects of system functionality, scalability of system mechanisms, and system ability to enact policy usage allocations. Full-scale testing and evaluation of the system in production environments is subject for future work.

The remained of this section discusses fairness in scheduling, differences between global and local fairshare scheduling, and relates this work to earlier efforts.

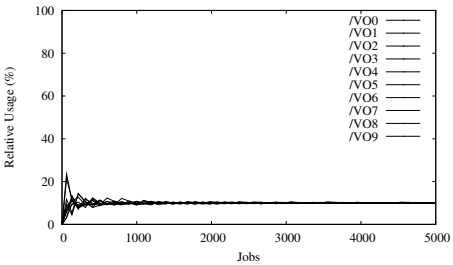


Figure 10: Convergence of total resource consumption for 10 different VOs. The system comprises a total of 1000 users symmetrically distributed with equal allocation shares over 100 projects and 10 VOs. Over 4 million jobs are run with stable convergence behavior. Illustration capped to region of interest (first 5000 jobs).

6.1. Global and Local Fairshare

FSGrid provides a framework for global (Grid-wide) fairshare scheduling. Existing schedulers (such as SLURM and Maui) contain fairshare prioritization mechanisms, but are designed for local (resource site) fairshare scheduling, using a single share policy, a common scheduler technology, and consider only local usage data. FSGrid offers a model for global fairshare where sites may have different share policies, use different schedulers, and operate on global usage data.

Local fairshare is often sufficient to manage job prioritization for HPC, as user identities are normally associated with a single site. In Grids, computational resources from several sites are aggregated into a common pool of resources available to all users of that Grid. The jobs of Grid users should be given the same prioritization regardless of which site in the Grid the job is submitted to, and the combined use across the Grid should be used for fairshare. For this, a global fairshare mechanism is required.

Although FSGrid is designed for global fairshare, the system can also be used for local fairshare job prioritization. Compared to local fairshare, global fairshare has additional challenges that include:

- Operation across administrative domains.
- Heterogeneity in technology, performance, availability, scheduling models, and allocation models.
- Greater usage data volumes.

- Usage and policy data updates has to be propagated to all participating sites, and each update may trigger a fairshare recalculation.
- Many different actors (site schedulers) depend on the same fairshare values simultaneously.

FSGrid may be deployed and configured in many different ways to suit individual environments. Any scheduler capable of calling Web Services can be integrated with FSGrid, and the policy model is based on site-specific local policies under full control of local administrators. To cope with Grid data volumes, fairshare values for known users are precomputed and cached. Similarly, as summaries of all usage for each slot in the time window are maintained at user level, usage updates only trigger recalculation of summaries for affected slots. Updated summaries are used in computation of fairshare values in following iterations.

Summaries and precomputed fairshare values are maintained at each FSGrid installation (normally one per site), and each scheduler can be served by a local FSGrid instance. As summaries are stored at each FSGrid instance, each site can have differently sized time windows (see Section 4.3) and purge usage data outside of the time window without affecting other sites.

6.2. Prior Work

This work builds on earlier efforts presented in [9], where preliminary versions of the policy model and simulations of the algorithm are presented. The main contributions of this work are a proposed architecture for realization of a decentralized system based on this algorithmic model, adaptations of the policy model and algorithm to facilitate distribution of the system, and a technical evaluation and analysis of the system. The architecture is designed for use in large scale environments, and focused on scalability through distribution and parallelization of data management and computations. Modifications of preliminary results presented in [9] include:

- Realization of the system. Prior work presented a simulation of the algorithm. This work presents a realization of a distributed system that is evaluated in an emulated environment.
- Extension of the fairshare algorithm with a framework for more fine-grained differentiation of resource consumption (fairshare operators).
- Increased precision of vector elements to allow a greater resolution in fairshare vectors.
- Reformulation of policy formats and interpretations to allow for dynamic updates of allocation policies.

7. Related Work

The fair Share scheduler [12] introduces the concept of user-level fair resource allocation in uni-processor sharing environments. The work introduces concepts such as fairness over time, support for different entitlements for different users, hierarchical policy structures, and sub-group isolation.

An evaluation of fair share in clusters or HPC systems is presented in [15]. Applicability of previous work on uni-processor sharing [12] to Grids or HPC systems is analyzed and simulated using logged data for thousands of real jobs. Effects of fair share on job prioritization are found to be small, partly because average system utilization is not high enough to cause enqueueing of jobs and partly because other factors (e.g. CPU requirements) are more deciding than differences in job priority.

Buyya et al. present a variation of the original FSGrid resource allocation strategy of [13]. Sub-groups (such as a sub-VO) may have dedicated resource allocations that can be used in conjunction with allocation of ancestor nodes. Consumption cost is used to select which allocation to use if several suitable alternatives are available. The aim is to maximize resource utilization, and fair allocation of resources between siblings in a hierarchy is not taken into consideration. An extension that also provides fair resource sharing is presented in [14]. Node job arrival rates are assumed to be known for all nodes in the system and the problem is formulated as a waiting time minimization problem. Jobs that cannot be immediately scheduled are rejected, and as jobs arrive with an assumed Poisson distribution, minimizing waiting time affects job acceptance rate. In [14] fairness is measured by job acceptance rate for different users.

Fair Execution Time Estimation (FETE) scheduling [2] is another take at Grid fair scheduling, where jobs are scheduled according to expected completion time as if running on a time-sharing system instead of a space-sharing system. Focus of this approach is to minimize risk of missing deadlines for submitted jobs.

Another hierarchical model presented in [8, 6], is used to allot resources from different sites to VOs and from VOs to users. Each sub-allocation includes both a burst allocation and an epoch allocation to control resource consumption in short- and long-term, respectively. GRUBER [7], is an architecture of this model that acts as a broker for resource usage Service Level Agreements (SLAs).

DI-GRUBER [5] extends GRUBER and adds support for distributed VO policy decision points. In these systems, VO polices are analogous to global policies in FSGrid and manage suballocation of resources within a VO. In contrast to FSGrid (where each site loads and enforces global policies), DI-GRUBER calls external decision points for VO policy decisions.

An evaluation of Grid resource allocation mechanisms is presented in [16]. Three different mechanisms considered are *volunteer*, *agreement-based*, and *economic* resource allocation. The agreement-based allocation mechanism used in the evaluation is based on earlier FSGrid work ([9]). The agreement based method was shown to have better overall resource utilization and suffer less degradation from high numbers of users compared to alternative approaches.

A comprehensive study on share scheduling mechanisms is presented in [3]. The study includes a thorough mathematical analysis of different strategies for share scheduling in uniprocessor, multiprocessor, and distributed systems.

More algorithms for fair scheduling in Grids are presented in [4]. The primary objective is to adhere to task deadlines. All tasks receive an equal share of resources regardless of number of jobs submitted. Excess resources not required by a task are divided equally among tasks that require more resources. Tasks may also be weighted to receive more than their equal share of available capacity.

A game-theoretic approach to fair Grid resource management is presented in [18]. This work considers the case where local scheduling decisions may be taken to optimize the system from the local schedulers point of view, and evaluates consequences of different levels of local scheduler autonomy in terms of (fair) scheduling.

Fair decentralized scheduling for Desktop Grids is presented in [1]. Fairness in this case is defined as minimizing the overhead of running each task on a shared infrastructure compared to a dedicated one. FSGrid defines fairness differently, and measures fairness as the difference between the expected and actual share of total resource consumption.

8. Future Work

A number of possible directions for future work are identified. Further investigation of trade-offs between FSGrid convergence parameters is expected to increase understanding of system behavior. Evaluation of impact of update latencies, usage decay functions, and job scheduling patterns are likely to influence parameterization and further development of the system. Evaluation of experiences from integration of the system in production use Grid deployments are expected to be of interest for further development of the system. Incorporation of scheduled and running jobs in fairshare job prioritization is likely to reduce impact of usage data update latency noise. Integration of the FSGrid job prioritization mechanism with additional cluster scheduling systems is expected to be of interest for system adoption.

9. Conclusion

In this work we present FSGrid, a decentralized system for fairshare-based Grid usage policy enactment built on three main contributions; a flexible policy model, a scalable fairshare calculation algorithm, and a decentralized architecture for parallelized fairshare prioritization of jobs. The system design is presented in detail, along with a performance evaluation and a discussion of the system.

The policy model supports mapping of VO structures onto policies, delegation of policy specification, and virtualization of usage credits. The fairshare calculation algorithm is self-adjusting and noise-stable, virtualizes resource site capacity, provides subgroup isolation within policy allocations, and adapts to changes in usage data and policy allocations. The architecture of the system is designed to facilitate decentralization of system deployments, precomputation and caching of scheduling data, and integrates non-intrusively with existing scheduling systems. The presented system can be utilized for job prioritization and scheduler-based policy enactment in Grid and HPC environments.

The performance evaluation illustrates the FSGrid policy allocation mechanism, demonstrates the feasibility of the approach, and identifies factors that manifest as noise in system convergence. The evaluation investigates trade-offs between convergence noise factors, and suggests how impact of these factors may be reduced. The discussion relates the proposed system to similar work and systems, and outlines the role of the system in Grid environments.

Acknowledgments

The authors extend their gratitude to Peter Gardfjäll for prior work, Lars Karlsson and Lars Larsson for feedback and discussions, and Raphaela Bieber-Bardt for work related to the project. The authors also acknowledge Tomas Ögren and Åke Sandgren for technical assistance to the project.

This work has been done in collaboration with the High Performance Computing Center North (HPC2N)

and has been funded in part by the Swedish Research Council (VR) under Contract 621-2005-3667, the Swedish National Infrastructure for Computing (SNIC), and the Swedish Government's strategic research project eSSENCE. The authors also acknowledge the Lawrence Berkeley National Laboratory (LBNL) for supporting the project under U.S. Department of Energy Contract DE-AC02-05CH11231.

References

[1] J. Celaya and L. Marchal. A Fair Decentralized Scheduler for Bag-of-Tasks Applications on Desktop Grids. In *CCGRID '10: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 538–541, Washington, DC, USA, 2010. IEEE Computer Society.

[2] E. Dafouli, P. Kokkinos, and E. A. Varvarigos. Fair Execution Time Estimation Scheduling in Computational Grids. In P. Kacsuk, R. Lovas, and Z. Nmeth, editors, *Distributed and Parallel Systems*, pages 93–104. Springer US, 2008.

[3] J. De Jongh. *Share scheduling in distributed systems*. PhD thesis, Delft Technical University, 2002.

[4] N. Doulamis, E. Varvarigos, and T. Varvarigou. Fair Scheduling Algorithms in Grids. *IEEE Transactions on Parallel and Distributed Systems*, 18:1630–1648, 2007.

[5] C. Dumitrescu, I. Raicu, and I. Foster. DI-GRUBER: A Distributed Approach to Grid Resource Brokering. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 38, Washington, DC, USA, 2005. IEEE Computer Society.

[6] C. L. Dumitrescu and I. Foster. Usage Policy-Based CPU Sharing in Virtual Organizations. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 53–60, Washington, DC, USA, 2004. IEEE Computer Society.

[7] C. L. Dumitrescu and I. Foster. GRUBER: A Grid Resource Usage SLA Broker. In J. C. Cunha and P. D. Medeiros, editors, *Euro-Par 2005 Parallel Processing*, volume 3648 of *Lecture Notes in Computer Science*, pages 465–474. Springer Berlin / Heidelberg, 2005.

[8] C. L. Dumitrescu, M. Wilde, and I. Foster. A model for usage policy-based resource allocation in grids. *Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on*, pages 191 – 200, jun. 2005.

[9] E. Elmroth and P. Gardfjäll. Design and evaluation of a decentralized system for Grid-wide fairshare scheduling. In H. Stockinger et al., editors, *First International Conference on e-Science and Grid Computing*, pages 221–229. IEEE CS Press, 2005.

[10] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[11] D. Jackson, Q. Snell, and M. Clement. Core Algorithms of the Maui Scheduler. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2221 of *Lecture Notes in Computer Science*, pages 87–102. Springer Berlin / Heidelberg, 2001.

[12] J. Kay and P. Lauder. A fair Share scheduler. *Commun. ACM*, 31(1):44–55, 1988.

[13] K. H. Kim and R. Buyya. Policy-based Resource Allocation in Hierarchical Virtual Organizations for Global Grids. In *SBAC-PAD '06: Proceedings of the 18th International Symposium on Computer Architecture and High Performance Computing*, pages 36–46, Washington, DC, USA, 2006. IEEE Computer Society.

[14] K. H. Kim and R. Buyya. Fair resource sharing in hierarchical virtual organizations for global grids. In *GRID '07: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, pages 50–57, Washington, DC, USA, 2007. IEEE Computer Society.

[15] S. D. Kleban and S. H. Clearwater. Fair Share on High Performance Computing Systems: What Does Fair Really Mean? In *CCGRID '03: Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 146, Washington, DC, USA, 2003. IEEE Computer Society.

[16] S. Krawczyk and K. Bubendorfer. Grid resource allocation: allocation mechanisms and utilisation patterns. In *AusGrid '08: Proceedings of the sixth Australasian workshop on Grid computing and e-research*, pages 73–81, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[17] R. Mach, R. Lepro-Metz, S. Jackson, and L. McGinnis. Usage Record - Format Recommendation, 2007.

[18] K. Rzadca, D. Trystram, and A. Wierzbicki. Fair Game-Theoretic Resource Management in Dedicated Grids. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 343–350, Washington, DC, USA, 2007. IEEE Computer Society.

[19] A. Yoo, M. Jette, and M. Grondona. SLURM: Simple Linux Utility for Resource Management. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 44–60. Springer Berlin / Heidelberg, 2003.