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

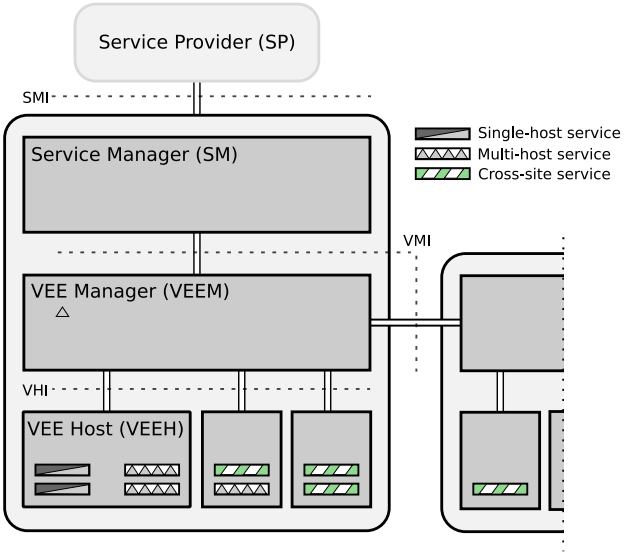

Fig. 1. The RESERVOIR architecture is made up of three different layers. The SM is the topmost layer, responsible for communicating with Service Providers and managing services on a larger scale. The VEEM layer is responsible for more fine-grained control over each service component, including placement 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 re-deployments.
- 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 back-end 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

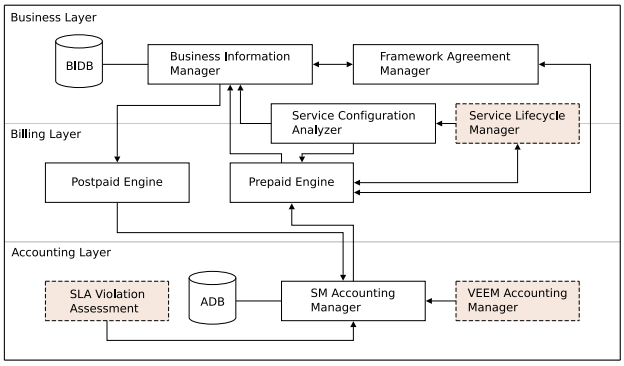

Fig. 2. Overview of the proposed architecture for accounting and billing within the RESERVOIR Service Manager. The figure shows the main components of the suggested architecture, with the connectors indicating the main interactions between components. Components with a dashed border in the figure handle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

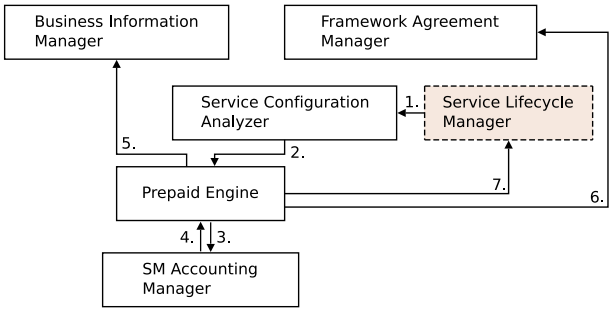

Fig. 3. The figure shows the procedure for the Prepaid Engine. The Prepaid Engine registers in the SM Accounting Manager to listen for updates when instructed to do so by the Service Configuration Analyzer. The Business Information Manager supplies the conversion from data concerning usage and violation 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.