A Highly Available Generic Billing Architecture for Heterogenous Mobile Cloud Services

Piyush Harsh
InIT Cloud Computing Lab
Zurich University of Applied Sciences
Winterthur, Switzerland 8401
Email: piyush.harsh@zhaw.ch

Konstantin Benz
InIT Cloud Computing Lab
Zurich University of Applied Sciences
Winterthur, Switzerland 8401
Email: benn@zhaw.ch

Irena Trajkovska
Dpto. Ingeniería de Sistemas Telemáticos
Universidad Politcnica de Madrid
Madrid, Spain 28040
Email: irenatr@gmail.com

Andy Edmonds
InIT Cloud Computing Lab
Zurich University of Applied Sciences
Winterthur, Switzerland 8401
Email: edmo@zhaw.ch

Paolo Maria Comi
Italtel S.p.A.
Innovation & Research
Castelletto, 20019 Milan Italy
Email: paolo.comi@italtel.com

Thomas Michael Bohnert
InIT Cloud Computing Lab
Zurich University of Applied Sciences
Winterthur, Switzerland 8401
Email: thomas.bohnert@zhaw.ch

***Abstract*—Rating, Charging, Billing (RCB) is the fundamental activity that enables a business to generate revenue stream depending on the resource consumption by their consumers. Traditionally, telecom operators have used custom designed, vertically integrated solution for RCB which often results in a complex system that is difficult to adapt to new service offerings. With telecom operator's desire to capitalize on cloud computing by using their vast amount of infrastructure, the need for a RCB solution that serves the needs of cloudified telcos is needed.**

In this paper we present an approach to implement a generic rating, charging, and billing engine that serves the business and technical needs of both cloudified telecom services and those of cloud service providers. Key to this is a generic accounting process to drive the design of the generic RCB architecture. We show how RCB as a service can be offered catering to not only traditional telco services, the new cloud services they wish and will offer, but packaged cloudified services to the consumers and application developers as well. Finally, we detail how our architecture can be distributed and key services replicated to ensure high-availability.

The end result of this paper is a solution that can enable telecom service providers to leverage the rapidly growing and accelerating cloud service market.

I. Introduction

In the telecoms' domain, the RCB process has been very tightly-coupled and vertically-integrated with their services. Therefore any new value addition (e.g. cloud services) on top of the offered service necessitates a complete overhaul of the RCB strategy, and many times technological ones, by the businesses. In this era of mash-ups and composed services, there is a real need of a completely generic RCB platform that can potentially support any composed service today and in the future.

We are conducting this research as part of Mobile Cloud Networking (MCN) project. MobileCloud goal is the convergence of the telecom and cloud worlds. Essentially it equates to: *Mobile Network* + *Decentralized Computing* + *Smart Storage* offered as one service based on cloud computing principles [1] e.g. on-demand, elastic, pay-as-you-go model.

Telecom services are normally offered over vertically integrated systems, comprising of Radio Access Network (RAN), Enhanced Packet Core (EPC), and IP Multimedia Subsystem (IMS). These services are supported by standards such as Diameter [2] and Radius [3] that provides Authentication, Authorization, and Accounting (AAA) support. More details of the current state of the art can be found in Section VI. With the emergence of smart-phones and always connected mobile devices, more and more value is created by mobile application developers on top of cloud services. Traditional telecom operators are being increasingly delegated to simply provide a dumb data pipe for such rich-experience mobile apps.

With a departure from traditional service models, the tightly integrated RCB solutions used currently, are rendered insufficient in dealing with the new models in MCN that will support dynamic service compositions using elements from both traditional telecom domain to be offered as a service plus elements of clouds offered as a service. Each composed-service being offered to the user (the application developer, or a Mobile Virtual Network Operator), can be offered by a single provider in its entirety, or individual services could be offered by independent operators.

This work plans to address this issue by providing an architecture that adapts to existing ones and models (Section IV) which help telecom operators embrace cloud computing principles and make an operator efficient through the use of clouds. In addition, MCN also aims at enabling new business models by extending the cloud, so that, an operator can provide customized bundled platforms comprised of cloud services and telecom features such as EPC to application developers. This would enable application developers to create next generation of fully integrated, rich mobile applications through custom provisioned app-development environments, customizing not just the traditional data-center elements, but also the elements of the telecom service stack.

And therein lies the motivation and need of a model for developing a RCB solution which is generic in nature so as to support requirements (Section III) of composed services in

a completely uniform manner. The proposed solution in this paper aims to be fully extendible in order to support new services that will be offered in the near-future.

Regardless of the nature of service offered, a business must conduct an internal accounting process in order to bill it's customers, and this process should be general across businesses and agnostic to the services offered. Hence in this paper we investigate how we can exploit this financial process for creating a completely generic rating-charging-billing model, as detailed in Section II.

With the this approach, RCB as a service can be offered to any generic service provider and support both the traditional monolithic service models, as well as new cloud-based atomic and composed service paradigm.

II. Accounting Process and Pricing Models

In order to comprehend the architectural design requirements on a generic RCB system, it is important to look into the overall accounting process and different pricing models that an organization could use in their billing process.

A. Accounting Process

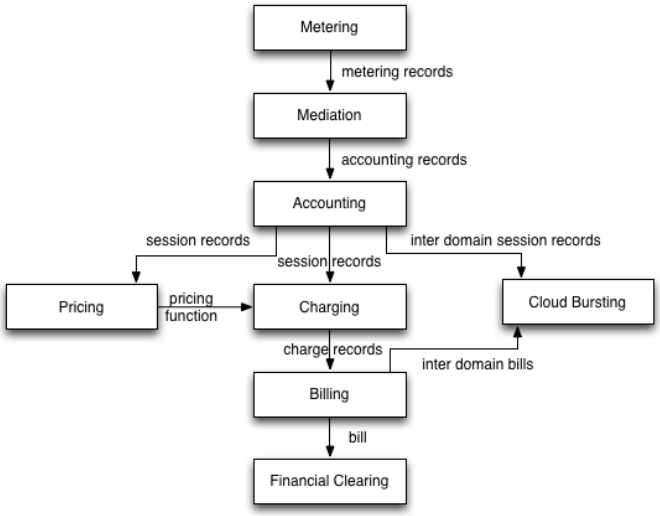

Fig. 1: General Accounting Process

In [4], the authors have captured the financial process for accounting cloud services. Figure 1 provides the overview of such an accounting process. It explains the general workflow and relations from the metering phase to the financial clearing process where the customer settles the invoice after the payment is processed. For our purpose, *we slightly adapted the concepts to support cloud bursting*. In our slightly adapted approach, various phases in the accounting process are -

- **Metering** - the process of collecting the various resource usage metrics of the consumers. This process is critical as without the raw metered data we can not properly customize our billing strategy. Without metering, businesses could offer their services essentially at a flat rate regardless of how high or low the customer's consumption is.
- **Mediation** - the process of assimilating and transforming the usage records that comes from different meters into a meter-agnostic format which could be processed by other modules in the accounting process cycle.
- **Accounting** - this part of the overall process is normally tasked with secured long term storage of accounting records generated by the mediation module, until at least the legally required timeframe. It also analyzes the accounting records and generates the session records for further processing. The stored accounting records come in handy in case of any billing dispute from the customers.
- **Pricing** - depending on the resource type, the pricing strategy will vary, e.g. - a provider may offer a flat rate for up-to 1 TB of storage, but the network bandwidth pricing could be based on the units of data sent/received. This function, depending on the resource type, outputs the appropriate pricing function to be applied to the accounting records.
- **Charging** - this is the process of applying the appropriate pricing functions to the accounting records to generate the charge records. Charge records contains the monetary value associated with the resource usage by the customer.
- **Roaming / Cloud Bursting** - This component is inspired by the roaming charges that one has to pay in the telecom domain. Similarly, if there is a cloud bursting scenario, then one has to consider that in the overall accounting process. This aggregation of billing information from external organization could be governed by special arrangements between providers. All these aspects can be handled at this phase in the overall process.
- **Billing** - this is the process of consolidating all the charge records since the last billing cycle. This stage also takes into account any discounts that were applicable in the cycle. Bills are generated for the customers as an output of this phase.
- **Financial Clearing** - generating bills is one aspect, sending the bills out to the customers and processing the payments through financial clearing houses is the main task supported in this phase of the financial process.

B. Pricing Models

In [4] [5], the authors also covered popular pricing models. In this section we summarize their findings. Pricing models are key in realizing an optimal revenue stream for the services being offered. The most common pricing models are[1] *time-based, volume-based, QoS based, flat-rate, Paris-metro model, priority-based, smart-market model, edge, responsive, proportional-fairness, cumulus, session-oriented, one-off and time-of-day based*. The correct choice of the pricing function for charging the resources could help differentiate one's service from the competition.

[1] for details please refer to the original study

Depending on the business scenario, one may have to adapt the generic pricing models. Some of the variations commonly used today are *free of charge, periodic-fees, discounts, pre-paid, online-accounting, offline-accounting, static-pricing, dynamic-pricing*, etc.

The generic RCB implementation, if to be used as a service by several customers, must be capable of supporting most of the pricing models and common variations used today. We will see later how our proposed architecture addresses the challenge.

III. Design Requirements

RCB system requirements in MCN are influenced by the general architectural requirements. The MCN architecture is service centric. Core telecom functions such as EPC, RAN, and BBU are offered as services. Some services have a built-in legacy "rating-charging" component, in which scenario, the proposed RCB architecture must utilize the charging data. In other cases the data format and message flows are to be designed in a completely service agnostic manner. Furthermore, RCB is the key process that leads to revenue generation, such a system should be highly available.

A. MCN Global Architecture

The MCN architecture follows a service oriented architecture. In the MCN architecture, all functional elements are modelled as services. The key architectural entities of the MCN architecture are:

- **Service Manager (SM)**: It provides an external interface to the user both programmatic and/or visual. It offers multi-tenant capable services to that user. The SM has two dimensions; the business which encodes business agreements, and the technical that manages the different Service Orchestrators of a particular tenant.
- **Service Orchestrator (SO)**: It embodies how the service is actually implemented. Generally, one SO per SM domain is instantiated per tenant. It oversees the complete (end-to-end) orchestration of a service instance (SI). It is implemented as a domain specific component and manages the service instance, which it creates, including scaling of the instance. The SO is managed by the SM and the SO monitors SI specific metrics related to the service instance. Although SIs are domain-specific, they are composed of service instance components (SIC).
- **CloudController (CC)**: Supports the deployment, provisioning, and disposal of SOs. To the SOs it also provides both atomic and support services through a Service Development Kit (SDK).

Below is a diagram of their relationships:

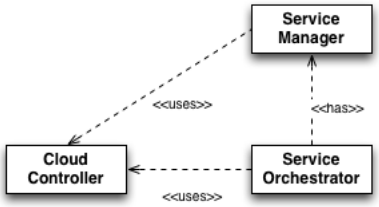

Fig. 2: Mobile Cloud Networking Architectural Entities and Relationships

Each architectural entity and service within MCN shares a common lifecycle model. The lifecycle model used in MCN is divided into two complementing phases, the business and the technical. For the business life cycle phase, the following stages are defined:

- **Design**: the service that will be offered is formulated and understood how it can be created from internal and outsourced services.
- **Agreement**: with a set of services identified, agreements related to service level agreements (SLA), pricing and access (AAA) can be entered with those service providers.

For the technical life cycle phase, the following stages are defined:

- **Design**: at this stage the service's technical design is carried out.
- **Implement**: with a service design the service is implemented. This entails the implementation of a SM and SO.
- **Deploy**: In order for the SM to take requests to create new service instances, the SO needs to be deployed using the CC.
- **Provision**: this phase is where the SO is instantiated and begins to create the services necessary to satisfy the SO's needs.
- **Runtime and Operation**: the SO has completed its job of providing the tenants service instance and is now monitoring and managing the service instance. It is during this step where scaling in and out of components is carried out.
- **Disposal**: the service instance's sub-components are destroyed and deleted.

To be integrated in MCN, the RCB architecture has to implement the SM and SO MCN architectural entities. As there is an existing CloudController within MCN, RCB as presented here can simply reuse it through the SDK.

IV. RCB Architecture

A high level RCB architecture is shown in figure 3. It showcases all the functional elements needed to handle different stages of a complete financial process. However it does not show in detail how the various elements of the overall architecture can be distributed and does not describe the communication interfaces between various modules.

The figure 3 shows OpenStack [6] and Ceilometer [7] monitoring as an example environment over which RCB could be deployed. The overall architecture is general enough to handle any service type as long as it can send necessary metrics data to the RCB service instance. The metric records from various services could be represented in any data representation

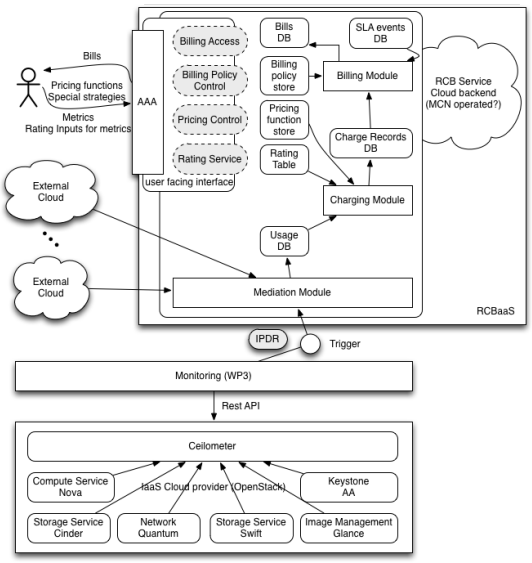

Fig. 3: Generic Rating, Charging, Billing Architecture

format standard. A likely candidate is IP Detail Records (IPDR) [8] standard.

In the overall architecture diagram, the various metrics taken from numerous (internal and external) channels come into the *Mediation Module*, whose task is to standardize the data format - translate from various supported data formats into a uniform format for other modules to consume.

The *Mediation Module* output i.e. the translated data records are then processed by an analytics engine (not shown in the overall architecture) to generate the usage records which are stored in the usage database for future retrieval and processing.

The *Charging Module* takes in a rating strategy and pricing function and processes the usage records to generate charge records. These charge records must be in a resource neutral format at this stage. The charge records could be generated periodically - as frequently as needed (configuration dependent) and stored in a secure database for future retrieval and processing by other modules.

The rest of the components' functionality is self-explanatory. The overall architecture shown is very easy to distribute. With a cloud service provider with several data-centers, the architecture can be split into two, collect usage and generate charge data locally at each data-center; collect the charge records from multiple locations, process and generate the bills in one data-center.

Since in MCN, RCB is to be provided as a service, we will present the design discussions from the implementation and deployment perspective.

A. Key Architectural Components

In this section we will describe key architectural elements that could be implemented as a standalone module which would interact with rest of the RCB architectural elements via secure message bus.

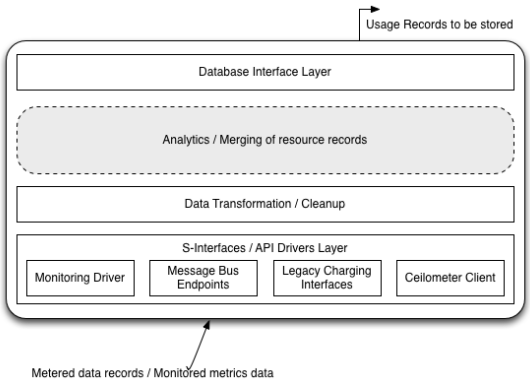

Fig. 4: Mediation Submodule

1) Mediation Submodule: Figure 4 describes in details the internal components of the mediation module. This module can be implemented as a highly available standalone service. The mediation module would be composed of -

- **S-Interfaces / API Drivers** - The southbound interface implements drivers for popular monitoring systems (Nagios [9], Ganglia [10], Zabbix [11], etc.) through which resource usage data can be filtered. It also implements the message-bus endpoints management for services that wish to send usage data directly to RCBaaS. Ceilometer is another optional client that could be supported.
- **Data Transformation / Cleanup** - The data coming through the southbound interfaces could be in disparate formats, they must be transformed in a common format for other modules to process in a uniform manner. They could be transformed into IPDR [8] records.
- **Analytics** - The monitored usage records in some situation needs to be combined together as part of a single user session. The analytics module analyzes the individual data records and performs the classification and statistical aggregation. The analytics engine can be implemented as an extendible engine where the users could supply their own analytics logic (ex. Datahero [12], Quantopian [13]).
- **DB-Interface** - Several popular data-store interfaces must be supported so as to provide flexibility with the choice of target store where the processed *usage records* could be kept for a configured time period.

2) Charging Module: The charging module uses the *usage record* from the *Usage Records DB* and applies the pricing function together with the rating strategy depending on the resource type to generate the *charge records* which is stored in *Charge Records DB* for future retrieval and analysis by other modules. Similar to *usage records*, the *charge records* are represented in a neutral, standard format, agnostic to the resource that resulted in such a record. This way the high-level modules are shielded from the low-level resources (ex. CPU, Disk, Network I/O, etc.).

Figure 5 shows the charging-module components. The *S - Database Interface Layer* connects to the *Usage Records DB*

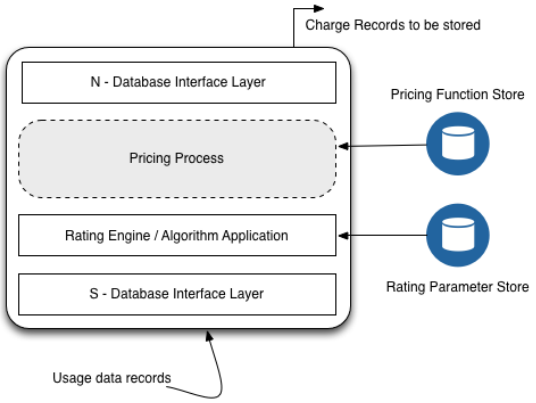

Fig. 5: Charging Submodule

to retrieve the data records for further processing. The *Rating Engine* governed by the rating process parameters / configuration values, processes the usage data and sends an intermediate data record to the *pricing process* for application of appropriate pricing function from the *Pricing Function Store*. The selection of the pricing function could be governed in-part by the *Rating Engine*. The *N - DB Interface* implements popular database drivers in order to send the *charge records* which contains the monetary value for the usage of a particular resource by the consumer, for secure storage in *Charge Records DB*. These records could be retrieved in future for further processing by other RCB modules.

3) Billing Module: Figure 6 describes the billing module that can be implemented as an independent package running on a separate node while interacting with other nodes using standardized interfaces.

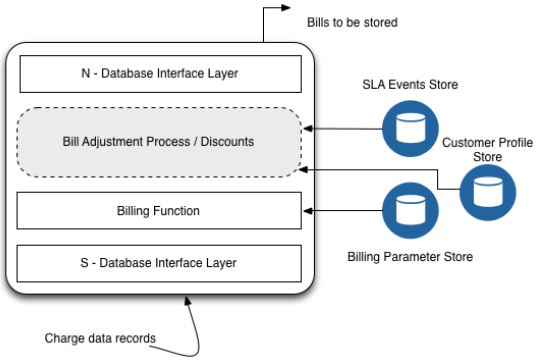

Fig. 6: Billing Submodule

The *S-Database Interface* implements the popular API drivers for connecting to the *Charge Records DB* and retrieving the data records from it. The charge records are aggregated by the *Billing Function* module, which simply generates the basic billed amount for various resources consumed. Depending on the individual consumer profile, the billed amount may need to be readjusted depending on pending discounts, penalties due to SLA violations, etc. This is taken care of by the *Billing Adjustment* process. The north-bound database interface implements popular database API drivers for storing the generated bills in a secure *Bills DB*.

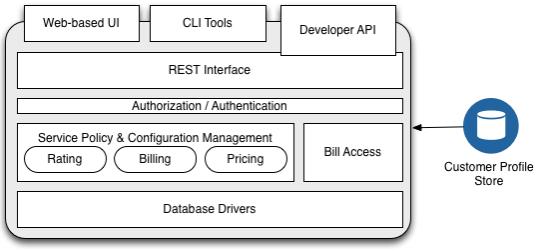

Fig. 7: User-Interface Submodule

4) User / Management Interface: Figure 7 shows in detail the user-interface module of the overall RCB architecture. It could provide multiple means of access to the service user: a web based UI, command line interface, and/or developers' kit in the form of an API - each built upon the underlying RESTful [14] interface. A standardized OCCI [15] billing interface could also be implemented to support interoperability. All user requests coming through the REST interface must go through authentication / authorization checks. Normally this module would allow service users to configure all aspects of the RCB process including policies and settings of rating engine, charging strategy, pricing model to be used for various resources consumed. It also allows them to access the generated bills to be forwarded to the collection centers or payment gateways. The interface presented to the service user would be governed by their profile settings.

5) Supporting Services: Authentication / Authorization service will be implemented as a cross module facilitator since every module in the RCB architecture needs proper authorization to talk to other modules of the service. There are numerous authentication and authorization solutions that could be utilized [16].

Individual modules deployed in a distributed environment needs a common messaging platform to synchronize the processes. Several open source messaging solutions (RabbitMQ [17], ZeroMQ [18], ActiveMQ [19]) could be utilized to implement the RCB messaging service.

Figure 8 shows the rating-charging-billing overall organization consisting of all supporting services and essential modules that could be easily distributed and made highly available if needed.

B. Strategies for RCB as a Service

Now that we have seen all the components of the generic RCB software architecture, how can it be used to support *as a service* concept? There are two possible strategies - individual instances per tenant which would require bringing up separate VM/OS-Container instances running all the above mentioned modules along with completely separate backend data-stores. In such a situation, the service user will have an instance isolated from other instances. The other solution would be to offer RCB instance by splicing the overall service. Different user's configurations and policies would be stored in the overall configuration and policy stores segregated using strict access control. The same would hold true for data-stores too. They

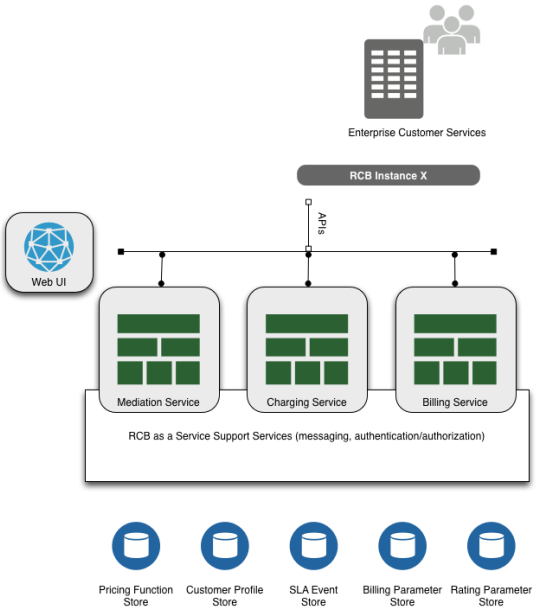

Fig. 8: RCB Overall Service Orchestration

could be offered out of the same database server or any data-store back-end to different users.

Whenever a new tenant is created, a management end point could be returned back to the user that would allow them to configure all the stages of the full financial process thus offering maximum level of tenant control. The service users could be billed in a numerous manner, ex. number of bills generated each month, metrics ingress rate, etc.

1) MCN Overall Architecture Alignment: The proposed RCBaaS fits nicely in the overall MCN service architecture. One would have to implement a SM representing the entry point of the RCB service. For each new tenant, a SO instance will be provisioned. The SO will then handle the deployment and run-time management of the RCB instance for that tenant.

V. Ensuring High Availability

The RCB overall architecture separates data from program logic. The data is stored in database files, stores and tables. The program logic is provided by software submodules, a web user interface and other control elements.

"High Availability" (HA) architectures exploit the fact that data is separated from program logic in IT processes. They make IT processes highly available by using a clustering technology (for increasing availability of data) and a distributed program logic (for automated failover). RCB can be turned into a HA system by clustering RCB data and by using a distributed program logic to control the RCB services.

HA clustering technology is based on replication and distribution of data on several redundant machine nodes (which form the cluster). In order to keep data consistent, it must be synchronized between all cluster nodes.

The distributed program logic is achieved by running distributed failover software over redundant computer nodes and by allowing the software to control the processes that run on each computer.

A. Degree of availability

Redundancy is the essence of High Availability. A non-redundant RCB architecture can not ensure high availability levels. If one of the RCB services fails, the whole RCB platform fails too. Though the usage of data replication and distributed failover software does enhance availability of the RCB system (compared to usage of non-redundant data and IT services), the actual degree of availability vastly depends on the number of cluster nodes and the configuration of the failover and data clustering technologies. At this point we must restrict our analysis to the very generic RCB High Availability architecture we see in figure 9. The architecture whose implementation produces a particularly high availability level can only be evaluated by exploration and tests of actual implementations. In the following subsections we want to describe the details of the generic RCB HA architecture.

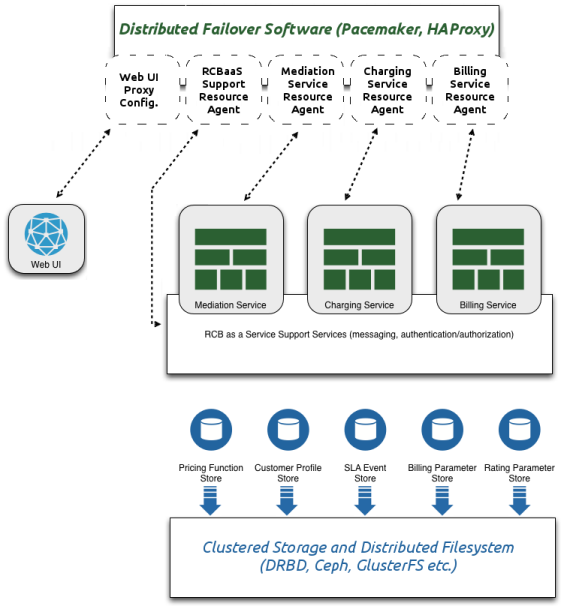

Fig. 9: Generic HA architecture for Rating, Charging, Billing

B. Clustering Technology

HA clustering technologies typically *use redundant storage devices* (e.g. disk partitions), *federate them into a single HA cluster* and *create some kind of distributed storage on top of the cluster*. The HA clustered devices are then threated by each node as if they were one single storage device. In order to access the clustered device through a single entry point, usually the cluster gets labelled with a "virtual" IP address. The virtual IP is a an IP address which is shared between cluster nodes and assigned to the cluster node which is currently actively managing the clustered device.

In order to make RCB data highly available, it must be stored on a HA clustered storage. Therefore all database files, stores and tables of the RCB must be transferred to the clustered storage. Then configuration files must be changed

in order to locate the data in the clustered storage. In the RCB HA architecture diagram (figure 9) the transfer of RCB data to a clustered storage is depicted with blue arrows.

C. Distributed Program Logic

While data can be made highly available by replicating it over several nodes, availability of a software program depends not only on redundant services, but also on management of the execution of those services. The RCB platform must be made highly available by *clustering all services which constitute the RCB platform* and then *run a distributed application which controls their execution*. This application should track the execution state of RCB services and grant access to the control flow of the RCB platform.

Program logic of the RCB application can be made highly available by installing identical RCB component services on multiple nodes and deploying a distributed failover software on all nodes. The redundant RCB services are then connected via "proxy configurations" or "resource agents" to the distributed failover software. The resource agents and the proxy configurations allow the distributed failover program to control execution of RCB services. The distributed failover application will check if an RCB service fails and process failover actions to recover from service outages. In figure 9 the connections between RCB services and resource agents or proxy configurations are depicted with dotted arrows.

Candidate technologies that allows us to create a HA RCB platform is covered in section VI.

VI. Technology Specifics, General Concepts and Related Work

A. Telecom and 3GPP

For wireless telecommunication charging is mainly covered by 3GPP specifications of 32.x series. These are 3GPP TS 32.240 "Charging architecture and principles" [20] and 3GPP TS 32.299 "Diameter charging application" [21]. The 3GPP standard supports both *offline* as well as *online* charging models. In offline charging, the resource usage is reported from the network to the Billing Domain (BD) after the resource usage has occurred. In online charging, a subscriber account, located in an online charging system, is queried prior to granting permission to use the requested network resource(s). 3GPP and Diameter approach to RCB is not suitable for a mash-up, composed service as the usage demands significant integration with the offered service which is time consuming. A more loosely coupled approach is needed which is offered by our model.

B. Research Trends in RCB

In this section we describe the existing approaches for RCB in the context of telcos, cloud providers and service providers. In the domain of the 3GPP telecommunication networks, a study by Grgic et al. [22] offers an extensive overview of the charging process. The authors propose: (a) signaling aspect, (b) inter-domain aspect and (c) service- and component-based aspect of online charging with respect to information utilization. They diagnose a lack of information specification and structuring, sharing issues and user privacy issues as research challenges for online charging systems with respect to the information access. A good report that classifies the pricing schemes for IP and ATM networks is presented in [23]. It analyses the current issues, advantages and disadvantages in the both pricing models and compares the pricing approaches. In his paper [24], Kelly describes a system model of charging, routing and flow control for broadband multiservice networks. The system assignees utility functions to the users and capacity constraints to the network. An example shows how the fairness criteria are associated with a particular utility function. The authors demonstrate that when users' choices of charges per unit time and the network's choice of allocated rates are in equilibrium, a system optimum is achieved. A new E-Charging API was proposed to Parlay and 3GPP OSA [25]. This API isolates the charging as a separate process offered by the Payment Service Provider and it is addressed for both the application service providers and the network operators. It permits the application service providers reach to the subscriber base of the payment service provider. Koutsopoulou et al. [26], propose a platform addressed for the next generation mobile network, for sophisticated and reconfigurable support of charging, accounting and billing process (CAB) as a discrete service. This platform reuses the existing network components according to the recommendations of the standardization groups. Apart from one stop billing, it supports separation of charging events based on transport, service and content usage. A set of APIs is provisioned for pricing related reconfigurations and deployment of charging services.

The emerging cloud computing market opens new possibilities for the telecommunication companies to maximize their revenue. In this context, Tselios et al. point out the closed-garden mentality of the telecommunication companies, their slow business model adoption and the lack of credibility to be a major handicap for cloud infrastructure adoption. A charging and billing layer is required, according to the authors, to capture the traffic records and provide the necessary charges to both departmental and individual levels. They conclude the urge for a new business model for increased price competition and improved customer service that will most likely enforce cloud adoption by the telcos [27]. The CGI group [28], identifies flexibility in billing as the missing link for cloud providers that would allow them to aggregate data and to understand usage patterns for better capacity planning and analysis of sales and marketing. We identified in the literature, an example of a system implementation of a model for deployment of different cloud business models based on the Internet Economics process [4]. The authors use jBiling as an accounting platform and IPDR protocol to fit different pricing schemes and tariffs, as well as better accounting on the usage of cloud services. Deelman et al. [29], take an approach towards using the cloud for science and analyze how a scientific application, given the availability of the clouds, can make the right cost-performance trade-off. In this context they study the cost of various workflow execution models and provisioning plans for cloud resources and prove the cloud to be a cost-effective choice for data-intensive applications. A monetary-based incentive for accounting and billing in Grid networks is presented in [30]. The novelty in this model is the support for multiple virtual organizations and multiple network operators. In addition, the authors present a Grid Economic architecture as a solution to the Large File Transfer

problem, that is essentially scalable, efficient and feasible over the Internet. Besides the standard AAA and Billing components, this architecture provides a Pricing, Metering and Security elements based on a price-wise or trust-wise service provisioned by the Grid node.

Two interesting applications of online charging are presented by Zuber [31] and Nagahara et al. [32]. The first one is a patented method for automatic tagging of documents and communications with filing and billing information for online social networks. This information can be further associated with each document and the communication can be customized to include categories most applicable to the business of the user. The second invention resolves the conventional charging system's limitation by facilitating children to use imaginary accounts for accessing on-line services such as: on-line shopping and video-on-demand. Finally Bhushan et al. [33], present a standardization-based work in B2B environment–a federated accounting management architecture for charging and billing.

The described approaches address particular segment or the entire RCB process, from individual provider's point of view. RCB paradigm is yet a novel service for cloud providers and therefore migrating this service to a higher level that will embrace heterogeneous providers and services is currently a challenging process. We have not registered so far a generic RCB solution aimed for composed services. What also distinguishes our approach in the RCB domain, is the consideration of high availability concept that is tightly coupled and highly important aspect for services' scalability.

C. Technologies for enabling HA

For highly available architectures, two type of enablers exist: technology for HA of data and technology for HA of software programs. A storage clustering technology is needed in order to make RCB data highly available. Typical examples of such HA clustering technologies are DRBD [34], Ceph [35] and GlusterFS [36]. The difference between these HA clustering technologies lies in synchronization of the clustered devices.

DRBD is a replicated copy of disk contents: after an initial synchronization of disk contents, disk writes can be performed by "primary" nodes only, and are propagated synchronously to all nodes [37]. DRBD is quite a simple and reliable mechanism, but because storage is rather copied than shared, DRBD storage does not scale very well.

Ceph is a more scalable solution, because (unlike in DRBD) consistency conditions can be relaxed and file writes need not be propagated synchronously to all nodes [35]. A major drawback of such relaxed replication is that a lookup service is needed in order to retrieve files and keep file data consistent [35]. Therefore Ceph separates file metadata from file contents.

An alternative to Ceph could be GlusterFS. Unlike Ceph, GlusterFS uses completely synchronous replication of files. Files are retrieved by a hash value which is assigned to them when they are written [38]. GlusterFS does not scale as good as Ceph, but it still scales better than DRBD, because the GlusterFS storage is not merely a copy of disks, but an abstraction of hardware devices [38].

The choice of the "right" clustering technology for RCB data depends on the requirements of the concrete RCB implementation: if a scalable solution is needed, Ceph is the "best" choice. If reliability is important, DRBD should be taken. If some compromise between scalability and reliability is needed (which is often the case in mobile networks), GlusterFS might be an adequate solution. RCB program logic can be made highly available by using a distributed failover software which manages the RCB services (e.g. the Billing submodule). Typical technologies which are capable of management of IT services are Pacemaker [39] and HAProxy [40].

Pacemaker is a distributed application which monitors "resources" (IT services) in a cluster and controls execution of these resources [39]. In contrast to Pacemaker, HAProxy is a HTTP/TCP load balancer which can detect service failures, failover unavailable services, and redirect user requests to currently available services [40]. HAProxy has the advantage that it couples user interactions to availability of IT services and transparently hides IT service outages to end users. A drawback is that it is mainly designed for HTTP/TCP-based applications. Pacemaker is more flexible than HAProxy: it can manage almost all possible kind of IT services. The major drawback of pacemaker is that it is not actively managing user requests.

As a HA solution for the RCB program logic, Pacemaker is suitable for management of core RCB services like Mediation, Charging, Billing and RCBaaS Support services. Pacemaker offers the flexibility which is required to manage those component services. For the Web UI service, it is better to use HAProxy because of its user request management and load balancing capabilities. Pacemaker and HAProxy have in common that they must be configured in order to observe availability of IT services and perform the required failover actions. In Pacemaker we need "resource agents" which tell Pacemaker how it can monitor, start or stop a particular service. In HAProxy a "proxy configuration" must be defined for each service which is monitored. For the connection of services with Pacemaker and HAProxy we plan to use custom resource agents for the RCB services (Mediation, Charging, Billing and RCBaaS Support) and a proxy configuration for the Web UI.

VII. Conclusion

In this paper we have provided a highly available, generic RCB service architecture that supports a standard accounting model. Every functional element of the proposed model is configurable, thereby making our solution work in any business environment. As a result, this solution is ideal to be offered as a service to different service providers. In section II, we provided an overview of accounting process to inspire the design. We presented a detailed architecture in section IV and analyzed and prepared a high-availability strategy in section V.

The next steps for us is to implement the architecture and integrate the solution over our OpenStack testbed. We will do more in depth study of different business models and check if our solution satisfies the RCB need of the control group.

Acknowledgment

The authors would like to thank the MCN consortium for providing the draft of the overall architecture diagrams which

helped them a lot to understand the RCB interaction with other components and in one way or other helped shape the RCB generic architecture. The work is supported by the European Community Seventh Framework Programme (FP7/ 20012013) under grant agreement no.318109.

REFERENCES

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," NIST special publication, vol. 800, p. 145, 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[2] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn, "Diameter Base Protocol," RFC 6733 (Standard Track), Internet Engineering Task Force, Oct. 2012. [Online]. Available: http://tools.ietf.org/html/rfc6733

[3] C. Rigney, "RADIUS Accounting," RFC 2866 (Informational), Internet Engineering Task Force, June 2000, updated by RFCs 2867, 5080. [Online]. Available: http://www.ietf.org/rfc/rfc2866.txt

[4] I. Ruiz-Agundez, Y. K. Penya, and P. G. Bringas, "Cloud computing services accounting," International Journal of Advanced Computer Research (IJACR), pp. 7–17, 2012.

[5] ——, "A taxonomy of the future internet accounting process," in Proceedings of ADVCOMP 2010 : The Fourth International Conference on Advanced Engineering Computing and Applications in Sciences. Florence, Italy: IARIA, 25–30 October 2010, pp. 111–117, iSBN: 978-1-61208-000-0.

[6] O. Community. (2013, Jul.) Openstack open source cloud computing software. [Online]. Available: http://www.openstack.org/

[7] O. Ceilometer Community. (2013, Jul.) Ceilometer - openstack. [Online]. Available: https://wiki.openstack.org/wiki/Ceilometer

[8] S. Cotton, B. Cockrell, P. Walls, and T. Givoly, "Network Data Management - Usage (NDM-U) For IP-Based Services. Service Specification - Cable Labs DOCSIS 2.0 SAMIS," IPDR Service Specifications NDM-U, Nov 2004.

[9] D. Josephsen, Building a Monitoring Infrastructure with Nagios. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007.

[10] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: Design, implementation and experience," Parallel Computing, vol. 30, p. 2004, 2003.

[11] P. Tader, "Server monitoring with zabbix," Linux J., vol. 2010, no. 195, Jul. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1883478.1883485

[12] DataHero. (2013, Jul.) DataHero. [Online]. Available: http://www.datahero.org/

[13] Quantopian. (2013, Jul.) Quantopian Community. [Online]. Available: https://quantopian.com

[14] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, 2000, aAI9980887.

[15] A. Edmonds, T. Metsch, A. Papaspyrou, and A. Richardson, "Toward an open cloud standard," Internet Computing, IEEE, vol. 16, no. 4, pp. 15–25, 2012.

[16] O. Keystone Community. (2013, Jul.) Keystone - openstack. [Online]. Available: https://wiki.openstack.org/wiki/Keystone

[17] J. Russell and R. Cohn, Rabbitmq. Book on Demand, 2012. [Online]. Available: http://books.google.ch/books?id=uO7IMgEACAAJ

[18] P. Hintjens, ZeroMQ: Messaging for Many Applications. O'Reilly Media, 2013. [Online]. Available: http://books.google.ch/books?id=TxHgtl_sFmgC

[19] B. Snyder, D. Bosanac, and R. Davies, ActiveMQ in Action. Greenwich, CT, USA: Manning Publications Co., 2011.

[20] "Telecommunication management; charging management; charging architecture and principles," 3rd Generation Partnership Project (3GPP), 650, route des Lucioles, Sophia-Antipolis 06921, France, Tech. Rep., 2013, TS 32.240, Rel-12. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/32240.htm

[21] "Telecommunication management; charging management; diameter charging applications," 3rd Generation Partnership Project (3GPP), 650, route des Lucioles, Sophia-Antipolis 06921, France, Tech. Rep., 2013, TS 32.299, Rel-12. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/32299.htm

[22] T. Grgic and M. Matijasevic, "An overview of online charging in 3gpp networks: new ways of utilizing user, network, and service-related information," International Journal of Network Management, vol. 23, no. 2, pp. 81–100, 2013. [Online]. Available: http://dx.doi.org/10.1002/nem.1816

[23] S. Bodamer, "Charging in multi-service networks," http://www.ikr.uni-stuttgart.de/Content/Publications/Archive/Bo_IB29_29702.pdf, 1998.

[24] F. Kelly, "Charging and rate control for elastic traffic," European Transactions on Telecommunications, vol. 8, no. 1, pp. 33–37, 1997. [Online]. Available: http://dx.doi.org/10.1002/ett.4460080106

[25] K. Luttge, "E-charging api: outsource charging to a payment service provider," in Intelligent Network Workshop, 2001 IEEE, 2001, pp. 216–222.

[26] M. Koutsopoulou, A. Kaloxylos, and A. Alonistioti, "Charging, accounting and billing as a sophisticated and reconfigurable discrete service for next generation mobile networks," in Vehicular Technology Conference, 2002. Proceedings. VTC 2002-Fall. 2002 IEEE 56th, vol. 4, 2002, pp. 2342–2345 vol.4.

[27] C. Tselios, I. Politis, V. Tselios, S. Kotsopoulos, and T. Dagiuklas, "Cloud computing: A great revenue opportunity for telecommunication industry," in FITCE Congress (FITCE), 51st, 6, Poznan, Poland, 2012.

[28] CGI, "Billing in the cloud: The missing link for cloud providers," http://www.cgi.com/files/white-papers/billing-in-the-cloud-e.pdf, 2010.

[29] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: The montage example," in High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for, 2008, pp. 1–12.

[30] S. Kotrotsos, P. Racz, C. Morariu, K. Iskioupi, D. Hausheer, and B. Stiller, "Business models, accounting and billing concepts in grid-aware networks," in Networks for Grid Applications, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, A. Doulamis, J. Mambretti, I. Tomkos, and T. Varvarigou, Eds. Springer Berlin Heidelberg, 2010, vol. 25, pp. 27–34. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-11733-6_4

[31] T. ZUBER, "Method for tagging documents and communications with filing and billing information," Patent Application, 05 2011, uS 2011/0106679 A1. [Online]. Available: http://www.patentlens.net/patentlens/patent/US_2011_0106679_A1/en/

[32] J. Nagahara, T. Nashida, H. Nakano, M. Niijima, Y. Sonoda, and Y. Kumagai, "Charging system in interactive on-line service," Patent, 04 2007, eP 0725376 B1. [Online]. Available: http://www.patentlens.net/patentlens/patent/EP_0725376_B1/en/

[33] B. Bhushan, M. Tschichholz, E. Leray, and W. Donnelly, "Federated accounting: service charging and billing in a business-to-business environment," in Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on, 2001, pp. 107–121.

[34] L. Ellenberg, "Drbd 9 and device-mapper: Linux block level storage replication," in Proceedings of the 15th International Linux System Technology Conference, 2008.

[35] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in Proceedings of the 7th symposium on Operating systems design and implementation, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=1298455.1298485

[36] E. B. Boyer, M. C. Broomfield, and T. A. Perrotti, "Glusterfs one storage server to rule them all," Los Alamos National Laboratory (LANL), Tech. Rep., 2012.

[37] F. Haas, P. Reisner, and L. Ellenberg, "The drbd user's guide," LINBIT HA Solutions GmbH, 2011.

[38] R. Hat, "Glusterfs: Red hat storage software appliance," 2011.

[39] M. Schwartzkopff, Clusterbau: Hochverfügbarkeit mit Pacemaker, Openais, Heartbeat und LVS. O'Reilly, 2010.

[40] V. Kaushal and V. Kaushal, "Autonomic fault tolerance using haproxy in cloud environment," International Journal of Advanced Engeneering Sciences and Technologies, vol. 7, 2010.