

Portable Cloud Appliance (PAChA)

WP1: Cost Optimisation Software

Final Report

Olivier Belli*, Charles Loomis[†], and Nabil Abdennadher*

*University of applied sciences, western Switzerland (HES-SO), hepia

Rue de la Prairie 4, 1202 Geneva, Switzerland

Email: {olivier.belli, nabil.abdennadher}@hesge.ch

[†]SixSq Sàrl

Avenue de France 6, 1202 Geneva, Switzerland

Email: cal@sixsq.com

Abstract

The cloud computing ecosystem comprises hundreds of providers, offering diverse computing services, incompatible APIs, and significantly different pricing models. Cloud application management platforms hide the heterogeneity of the services and APIs, allowing, to varying degrees, portability between providers. These tools remove technical barriers to switching providers, but they do not provide a mechanism for evaluating the cost effectiveness of switching.

This report summarises the research work carried out in the context of WP1. It presents a Decision Support System (DSS), working within the SlipStream cloud application management platform. THE DSS evaluates the costs of a customer's applications using resources from different cloud service providers. The system (1) generalizes and normalizes multiple cloud pricing models and (2) gathers pricing data from cloud providers. These, in conjunction with the application resource consumption model, allow the cloud pricing module to estimate a price for the application for each cloud provider.

Index Terms

cloud, cloud pricing, IaaS, comparison, cost.

I. INTRODUCTION

During the last few years, cloud computing has grown to become a standard and cost effective way to supply IT resources. The cloud computing ecosystem has grown enormously as numerous new cloud providers have joined the existing giants in the sector: Amazon, IBM, and Microsoft. All of the providers, old and new, offer similar Infrastructure-as-a-Service (IaaS) resources, differentiating themselves with a varied array of higher-level services.

However, there is no standard way to monetize cloud resources [1]. The price of running a cloud based application depends on many factors, including the number of servers, storage, bandwidth usage, temporary servers for peaks in demand, and execution time [2]. Consequently, the pricing model, even for the IaaS resources, varies greatly from provider to provider.

This report presents a cost-optimized cloud application placement tool designed within the PAChA project (WP1). The tool is based on a Resource Consumption Model (RCM). The ultimate goal is to optimize the placement of an application based on price. The tool addresses two questions:

- How to retrieve pricing data and metrics used to create the RCM of any cloud application?
- How to use this RCM to compute the application price on each cloud provider?

The RCM must deal with (1) the heterogeneity of the cloud providers pricing models and (2) the “diversity” of metrics used to calculate the deployment cost of cloud applications.

Optimizing the placement of cloud application based on price, the ultimate goal, requires several stages:

a) Collecting and Normalizing Pricing Data: Computing the cost of a cloud application requires the pricing data of all cloud providers. This task is far from trivial as APIs providing pricing data are not common; a workaround must be found. Once the pricing data is available, it must be normalized and stored in a common generic model to avoid complications from provider-specific pricing models with differing semantics.

b) Building the Resource Consumption Model: This stage consists of creating a resource consumption model (RCM) for a particular cloud application. The idea consists of identifying and quantifying the cost-relevant metrics of the cloud application. Assessing these metrics can be done manually or with a cloud application management platform. The RCM is built from these metrics and from the normalized pricing data.

c) *Computing the Cost*: The last stage is the submission of the resource consumption model to a cost computing algorithm (CCA). The CCA computes the execution cost of a given application (RCM) for each cloud provider. This final step allows costs to be compared between providers and gives a cost ranking of the providers to the user.

For two reasons, this report focuses on services that provide IaaS resources. First, these services are available from the majority of cloud providers, while higher level services are both more provider-specific and more varied (SaaS and PaaS services are mostly complex cloud applications). Second, cloud application management platforms usually use the common IaaS services. The desire to demonstrate optimized placement through a cloud management platform limits the scope of this report.¹

It is also important to point out that this report only treats the cost of an application: criteria such as performance [3] [4], Service Level Agreements (SLA) [5], legal dispositions, or higher-level service availability [5] are out of the scope of the PACHA project.

The rest of the report is organized as follows. Section II presents previous work related to cloud pricing. Section III describes the global architecture and the components of the Cost-Optimized Cloud Application Placement Tool (COCA-PT) developed within the PACHA project. Section IV presents the integration of COCA-PT with SlipStream, a multi-cloud application management platform. Section V discusses experimental results (costs) obtained when deploying applications on cloud infrastructures using COCA-PT. The last section deals with the limitations and future directions of this research.

II. RELATED WORK

Benedikt Martens *et al.* [2] present a method to compute the Total Cost of Ownership (TCO) of cloud computing services from the customer's perspective. They provide a mathematical approach to compute the TCO after identifying all of the costs and types of costs. Their approach assumes that consumers already have their own IT infrastructure (computers and Internet connection) and that their goal is just to compare public clouds, not private vs. public cloud. We use this as a base to identify the required metrics to build the RCM.

Ang Li *et al.* [3] focus on comparing cloud providers. While the previous paper mainly addresses pricing, this paper also addresses performance. They provide CloudCmp, a tool that helps customers select a provider. They base their performance comparison on three types of widely used applications: "storage intensive e-commerce web service", "computation intensive scientific computing application" and "latency sensitive website serving static objects". Their tool is presented as fair to the providers and helpful in identifying the providers' weak services. This addresses part of our problem but does not cover the dynamic monitoring and comparison of live applications that we cover here.

Silvadoon Chaisiri *et al.* [7] study the different provisioning plans—on-demand and reservation—in cloud computing. Their Optimal Cloud Resource Provisioning algorithm aims to minimize the cost by choosing the most suitable plan. This algorithm takes into account the demand and the price uncertainty to propose the optimal plan when provisioning virtual machines. As we focus on live optimization, we do not currently consider reservation plans; however, this is a factor to consider in the future for long-lived applications.

Hong-Linh Truong and Schahram Dustdar [6] study the deployment of scientific applications. They determine if those applications are optimally deployed on a public cloud, a local infrastructure, or a mix. Their analysis includes both requirements and cost; hence, they provide models to estimate cloud costs.

Gabriella Laatikainen *et al.* [1] study the different pricing models of many IaaS, PaaS, and SaaS cloud providers. Their work is based on the idea that a condition for success is a "well-defined pricing strategy". They point out that despite the heterogeneity and complexity of the models, common patterns exist. They apply the "Scope, Base, Influence, Formula, Temporal Rights" pricing model to the cloud context to derive their adapted pricing framework. All of the papers, and especially this one, demonstrate the need for a normalization strategy for pricing data.

We extend the above ideas to create a live, automatic, and general cost comparison process for all types of cloud applications.

III. COST-OPTIMIZED CLOUD APPLICATION PLACEMENT TOOL

The Cost-Optimized Cloud Application Placement Tool (COCA-PT) described in this section is one part of a decision support system that assists users in selecting cloud providers for their applications. Unlike the majority of the projects presented in the previous section, the COCA-PT supports all types of applications. From a high-level perspective, two categories of applications are deployed in the cloud:

- A **service-oriented** application is persistent: it has no stopping point. This category contains applications such as web sites (servers, databases, load balancers), and application servers.
- A **task-oriented** application is well defined in time. It has clear starting and stopping times. An application is launched to perform a task and is destroyed on completion. This category contains, for example, scientific computation applications and video rendering applications.

¹The design of the cost computing algorithm and the resource consumption model is generic and should work equally well with higher level services, such as database services.

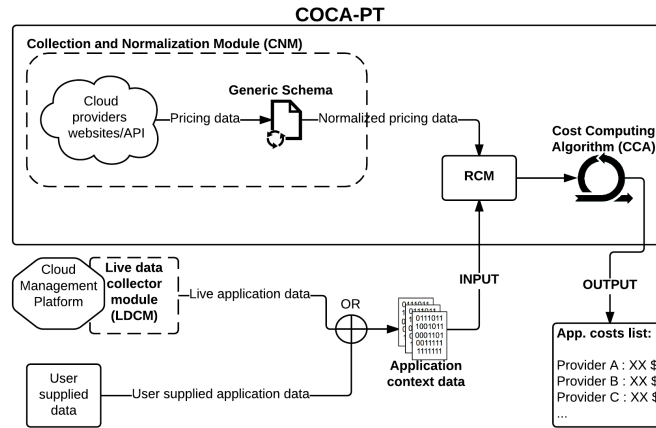


Fig. 1. Full Cost Comparison Process

COCA-PT is designed to support both types of applications throughout the full lifecycle of an application. The very different time profiles of these types of applications raise two problems for us. First, prices will change over time, impacting especially long-running applications. Because of this, pricing data will have to be retrieved frequently to ensure accurate price estimations. Second, cloud providers often offer discounts and preferential rates based on volume or extended use. These discounts (which are often undocumented or difficult to find) must be applied to provide accurate recommendations to users.

COCA-PT is composed of two modules: the Collection and Normalization Module (CNM) that collects and normalizes pricing data from providers and the Cost Computing Algorithm (CCA) that estimates the cost of an application on the cloud. Using data from the CNM and provided usage data, a Resource Consumption Model (RCM) is built and given as input to the CCA. The output of the system is a list of costs for different providers. Figure 1 shows the full cost comparison process and the relationships between the components. The components are described in detail below.

A. Pricing Data Collection

Cloud providers supply APIs for nearly all aspects of their services; pricing data, however, is a notable exception. Amazon Web Services (AWS), the largest cloud provider (31% market share in 2015 [10]) only supplied a price list API starting in December 2015 [8]. Google Compute Engine (GCE) offers the price list used by its pricing calculator, but without any accuracy guarantees. In the future, other providers may follow the lead of AWS and GCE, but for now, other methods must be used to collect pricing data for most cloud providers. Further complicating the situation, pricing data frequently change (sometime daily), requiring constant monitoring of the prices.

Given these constraints, we opted for a semi-manual approach for pricing data collection. Pricing data is collected directly from the providers' websites to ensure accuracy and the process is automated to make frequent updates possible. For each cloud, an analysis of the pricing model is conducted. This analysis identifies information that is not explicitly given by the pricing web pages: discount methods, billing periods, smallest time unit billed, etc. Once the model is understood, a web page parser for the cloud provider is written. This parser scrapes information from the relevant web pages and combines this information with the tacit model information.

One of the drawbacks of this solution is the fact that the parsers must be re-adapted when the cloud providers change the structure of their pricing web pages. Given the current API situation, it is not possible to do otherwise. Web parsing can be replaced by API calls as pricing data APIs become available. Using "intelligent" techniques to automatically find targeted data on web pages is also a possibility. This possibility has been ignored for two reasons: it is easy to re-adapt a web parser and we believe APIs will be provided in the future.

B. Normalization

Collected pricing data do not have the same format nor the same semantics. They have to be normalized in order to be exploited. This section describes how we developed a generic schema (Figure 2) that fits multiple cloud pricing models (multiple charging/discount methods) and different types of services.

The schema is based on the analysis of pricing models from a number of popular cloud providers: AWS [12], GCE [13], Exoscale [14], Ultimum [16], Rackspace [17], SoftLayer [18] and others. The main differences between them are how discounts

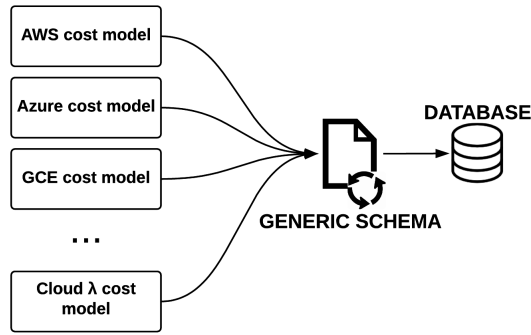


Fig. 2. Cost Model Normalization

are applied, the time for charging, the resources they propose (e.g. fixed-size vs. custom virtual machines and disks), and their free services/tiers.

The schema supports pricing for four main cloud services [3].

- Compute Instances: Virtual machines running in the cloud. The cost usually depends on the instance type and the time of use.
- Storage Disks: Fixed or dynamically-sized disks attached to virtual machines. The price usually depends on disk size and time of use.
- Networking: Network traffic into and out of virtual machines or other cloud services. The price usually depends on the monthly network traffic between the cloud components and the Internet.
- Object Storage: Document/object storage service. The price usually depends on the number of requests made to the service, the quantity of data stored, and the data served over the network.

The schema does not claim to be universal, but it is flexible enough to support most cloud providers. It is composed of a set of attributes with defined semantics and data types. In the interest of reuse and interoperability, some parts of the schema are inspired by Schema.org [23]. The schema does not follow the AWS or GCE APIs, primarily because our solution must be more generic and must support different billing/sampling time configurations and services. Table I contains the schema definition.

This schema is used to represent a cloud resource (an element of a cloud service) and is part of the Resource Consumption Model.

C. Application Context

A cloud application is defined by a set of cloud resources and software. The “context” of a cloud application describes the “conditions” under which the application is deployed and executed. This information can either be supplied manually by a user (usually through application profiling) or automatically by collecting live data from the cloud and/or the application itself.

To illustrate this, consider a simple example. The example application consists of a single virtual machine with 2 vCPU and 7 GB RAM. On AWS, it would use a *t2.large* instance (2 vCPU, 8 GB RAM, 0.104 USD per hour, no discount). On GCE, it would require a *n1-standard-2* instance (2 vCPU, 7.5 GB RAM, 0.100 USD per hour, sustained usage discount). This application allows clients to connect to the instance and download a file.

Consider two contexts. In context A, the application exposes a 0.1 GB file and serves few clients per month (100). In context B, the application exposes a 1 GB file and serves more clients per month (1200). Table II displays the price per month for GCE and AWS for both contexts². The results take into account discounts: the GCE sustained usage, network discounts, and free quantities. As shown in the Table, the principal part of the cost shifts between the network and VM cost depending on the context. As a consequence in this case, the preferred provider changes between AWS and GCE. This shows that a cloud application’s cost is tightly coupled to its context.

D. Resource Consumption Model (RCM)

The RCM defines the cloud resource usage of a given cloud application under a certain context. The RCM contains the quantities of consumed resources in one structure. The resources include:

- VMs: Number, type, time of usage etc.
- Network: Ingress/egress, Internet/local traffic

²Based on May 2016 prices assuming 730-hour months.

TABLE I
PRICING SCHEMA ATTRIBUTES

Name	Description
resourceType	The resource type that is being priced, for example, "vm" or "storage".
name	The name of the resource, for example "t2.micro" or "network.egress".
descriptionVector	Vector used to describe the resource. For example, an instance with fixed resources would be described by {"vcpu": 2, "ram": 4, "disk": 10}. If the resource is not fixed, this vector is empty.
price	The price per unit of a cloud resource (e.g. VM, network, disk).
priceCurrency	The currency for the price. This follows the 3 letter currency code from the ISO 4217 standard [21].
unitCode	The UN/CEFACT 3 letter code [22] for the units of the resource.
billingTimeCode	Value describing the billing period for the item. Note that this value also indicates when to reset the discounts. See the UN/CEFACT 3 letter code [22] describing time periods.
sampleTimeCode	Smallest billing unit. See the UN/CEFACT 3 letter code [22] describing time periods.
discount	A list of attributes describing the method for applying discounts on the resource.
discount.method	Method to use for calculating the discount. Current supported values are "Progressive", "Reservation", and "None". A progressive discount is applied with sustained use; the more a resource is used in a billing period, the less it costs.
discount.steps	Additional information for calculating a pricing discount. This is an array of pairs of numeric values that depend on the charging method indicated by the discount method, unitCode and time codes. The values are the threshold and the associated discount.
freeQuantity	The threshold below which the use of the resource is free. The default value is zero.
associatedCosts	An array of associated costs described by the same schema. This attribute is used to describe more complex resources that are priced by multiple factors (e.g. storage + number of I/O requests).

TABLE II
EXAMPLE APPLICATION PRICE (USD) COMPARISON

	AWS		GCE	
	Context A	Context B	Context A	Context B
Instance cost	75.92	75.92	51.10	51.10
Network cost	0.81	107.91	1.20	142.24
Total	76.73	183.83	52.30	193.34

- Disks: Number, size, time of usage etc.
- Other Services: Object Storage etc.

Figure 3 shows a simple RCM for a cloud website application composed of a database, 2 web servers and a load balancer. The 4 VMs require an additional disk each; the egress network consumption is also present in the RCM. The same application under a different context would have a different RCM. Note that this RCM figure is simplified and adapted for illustration purposes.

E. Cost Computing Algorithm (CCA)

The costs of running an application on the cloud depends on resources usage. These resources can be classified in two distinct categories: infrastructure resources and operating resources.

1) *Infrastructure Resources*: Infrastructure resources are not context-dependant. To illustrate, consider a simple load-balanced website application. This application is composed of 4 VMs: a load balancer, a database VM and two web servers. The infrastructure resources would be the 4 VMs (load balancer, 2 web servers, 1 database) and the associated storage. These

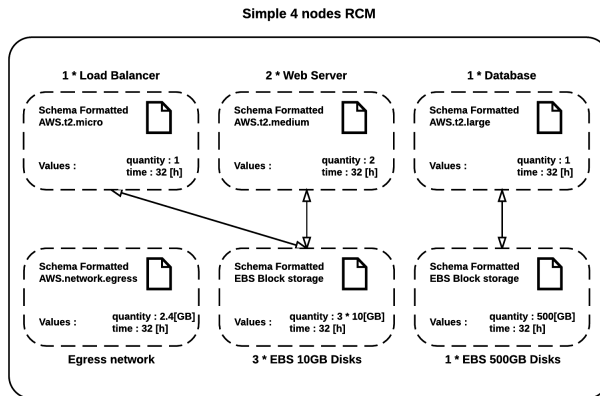


Fig. 3. Resource Consumption Model

resources will not change depending on the usage (for a fixed time period); data are gathered from the infrastructure. To summarize, these resources comprise infrastructure VMs and infrastructure disks. When using a cloud application management platform, infrastructure resource consumption is quite easy to collect because the platform allocates and releases these resources itself.

2) *Operating Resources*: Operating resources are more varied and much more difficult to assess. These resources are composed of:

- (1) Temporary VMs: Extra VMs are started or stopped as an application's requirements change over time. For example, one can add servers to maintain the quality of service during peaks in usage or nodes can be started for the compute-intensive stages of a scientific application.
- (2) Temporary Disks: Extra disks are added or removed depending on the current requirements of the application.
- (3) Network/Internet Traffic: Network traffic depends on the nature of the application and the behavior of its users.
- (4) Other Service Metrics: Usage measurements for many other service are possible. A common example is charging based on the number of requests to the object storage service.

Resource consumption related to temporary VMs (1) and disks (2) can be calculated with a process similar to that used for infrastructure resources.

The network part (3) is more problematic. Cloud providers usually only provide this information periodically with bills and then only in the aggregate, without per-VM (or per-service) granularity.

One solution is to include an additional, router VM that monitors all the traffic between the "local" network and the Internet. Although feasible, this solution has two flaws: it increases the price of the application and it creates a major network bottleneck. Using the smallest possible VM would minimize the cost overhead but increase the performance penalty because the smallest VMs come with very limited network bandwidth capabilities.

The second solution consists of embedding a small monitoring agent in all VMs belonging to the cloud application. The monitoring agent informs the cloud application management platform of the VM's network consumption (quantity, destination, etc.). As for the previous solution, monitoring is not "free" as the agent consumes CPU and memory resources on the VM; the agent could also be stopped (inadvertently or intentionally) by the user. On balance, this second solution is preferred.

Usage metrics for higher level services (4) are problematic because most of them are impossible to monitor from outside of the cloud infrastructure. As an example, consider object storage. There are usually three components to the cost:

- Network: Downloading objects from the Internet
- Storage: Size of the stored objects
- Requests: Number of requests handled by the object store

Of these, only the storage size is possible to monitor. Object storage network traffic, unlike VM network traffic, cannot be directly monitored because it is not possible to add a monitoring agent to the object store itself. Traffic from the application itself can be monitored indirectly through the application's VMs. However, direct access to the store from outside clients cannot be monitored unless the cloud provider itself provides an API for this. The same argument applies to request metrics. Because

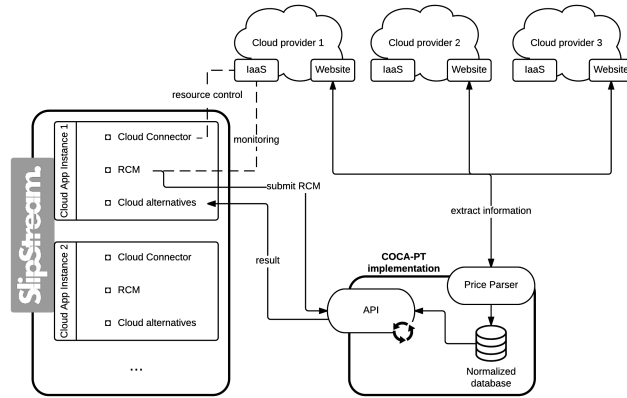


Fig. 4. Implemented System

of these limitations, the resource consumption model of an application can only be automatically created for lower-level, IaaS resources.

IV. SYSTEM IMPLEMENTATION

The implemented system contains two components:

- The Cost-Optimized Cloud Application Placement Tool (COCA-PT) and
- SlipStream [11], a cloud application management platform. SlipStream is used to automatically collect live data from the cloud and assess the "context" of the deployed application.

COCA-PT communicates with the cloud providers' websites. It collects price information via a web scraper or an API (if available). The module is responsible for collecting pricing information and making the cost computing system available through an API.

SlipStream is a multi-cloud application management platform supporting the system. Its users can define cloud-independent applications and deploy them easily to different cloud infrastructures. It facilitates automated deployment through synchronized communication between application components. SlipStream manages the full lifecycle of cloud applications, including the possibility to scale running applications. SlipStream produces "events" for all lifecycle state changes. These events allow the production of the resource consumption model, which is then used to build the price comparison list via the COCA-PT API when the infrastructure data (III-E1).

Figure 4 illustrates the complete, implemented system. The SlipStream platform communicates with the different cloud IaaS APIs to manage applications. The platform uses a Live Data Collector Module (LDCM) to profile applications' resource usages. Resources usages are transferred to COCA-PT to combine those with pricing data to build RCMs and compute costs on different providers. Each component is described in more detail below.

A. Pricing module

COCA-PT is composed here of three elements.

- Price Parsers: These are responsible for retrieving the pricing information from the cloud websites. They parse³ the pricing web pages to extract pricing data and format that data according to the generic schema (Table I). Parsers run on a regular basis to ensure that the database is up to date.
- Database: The extracted and formatted pricing data is stored in the database.
- API: The API exposes the database elements, provides resource matching, and computes the costs using the CCA.

B. Multi-cloud Application Management Platform

The platform used to manage applications, VMs, monitoring etc. is SlipStream [11]. The relevant features are:

- Defining multi-node, cloud-independent applications
- Starting and stopping predefined applications on different cloud providers

³If APIs are available, the APIs are used in preference to parsing the HTML of web pages.

- Synchronizing the configuration of applications
- Scaling cloud applications (e.g. adding or removing nodes on the fly)
- Executing code on VMs automatically
- Providing a REST API

Using these features, we can create the resource consumption model (RCM) of running applications. To reach this goal, the following steps are required.

1) *Events Analysis*: Using the API, we are able to retrieve the events created by SlipStream for lifecycle state changes. The relevant events concern the provisioning or removal of VMs or disks. With those, a part of the resource consumption model can be built. The data retrieved here are mostly *infrastructure data* (III-E1), but also data for temporary VMs and disks from the *operational data* (III-E2) are accessible.

2) *Network Monitoring Agent*: To complete the resource consumption model, networking data are required. The approach here is to develop a small agent able to quantify and identify the network traffic. The source and the destination of the network packets must be determined, as these factors impact the price. The agent communicates its data directly to SlipStream. The footprint of the agent on the VM's performance should be as low as possible.

3) *Modified Applications*: Applications are modified to test multiple scenarios and to add the network agent to each VM.

Using this data, the resource consumption model can be built. Submitting the model to the pricing module API returns a list of cloud providers alongside the price the application would cost on each provider (in the current utilization context). The utilization context should always be representative in order for the system to perform well. Using the cost/provider list, the consumer can easily determine if the application would be cheaper on another cloud. As described before, the ranking is only based on the price; users have absolutely no guarantees that the quality of service or performance will be the same on the new cloud.

COCA-PT has been tested with multiple providers. Different strategies were used to validate the results, depending on the provider. One difficulty with the comparison is that most cloud providers do not give access to detailed, hour by hour, component by component billing. The bill is often accessible at the end of every month and generally presents the consolidated, aggregated costs. Achieving precise comparisons with these bills is very difficult. An exception is the Swiss cloud provider Exoscale: its detailed billing allows a direct validation of the pricing module for that platform.

Many clouds provide cost simulators; these allow another method for validating the pricing module. This method was used for Amazon Web Services, Google Compute Engine and Azure.

To validate the pricing module, we used several scenarios consisting of distinctive applications under different contexts. In the simulated approach, we defined a cloud resource consumption for each scenario. These resource consumptions were submitted to the simulators and to the pricing API. The comparison between results showed no difference.

The practical method was tested with Exoscale with the following applications and results:

- 1) A single VM task-oriented application. This application was executed with different parameters and lasted between 8 hours and 3 days. The difference between the estimated cost and the real cost was under 1%.
- 2) A single VM web server application under specified traffic load. This application was executed for 2 weeks. The estimated cost and the real cost differed by less than 1%.
- 3) A multi-VM web site application with traffic load variations with scaling. This application was executed for 2 weeks. The load was varied to ensure that the auto-scaling took place. The estimated cost and the real cost differed by less than 2%.

Like the simulated approach, the practical approach showed only minor differences between the actual, billed charges and those calculated by the pricing module.

The practical approach must still be used with other providers and scenarios to be able to quantify the accuracy of the system.

V. PRICING ALGORITHM VALIDATION

The aim of this section is to present the experiments carried out to test and validate the COCA-PT framework. two types of experiments were conducted:

- Validation using calculators: This experimentation is straight-forward. Some cloud providers propose pricing calculators to assess the cost of launching services on their cloud (i.e. Amazon [25], Google [26], Microsoft [27]). We can then compare this assessment with the cost provided by COCA-PT. This approach has three main disadvantages: (1) most of providers don't offer a pricing calculator, (2) calculators only offer estimations and (3) this approach is theoretical, it does not use real data.
- Validation using usage bills: The idea is to deploy a given application with a given context on a given Cloud infrastructure and compare the cost provided by COCA-PT with the amount of the cloud provider's bill.

It's worth reminding that the results presented here are from Feb. 2017. Running the same tests today might give different results.

TABLE III
SERVICES AND CONTEXTS

Service type	Name	Contexts
Instance	I1	a: 1h usage b: 1/2 month usage c: 1 month usage
Instance	I2	a: 1h usage b: 1/2 month usage c: 1 month usage
Instance	I3	a: 1h usage b: 1/2 month usage c: 1 month usage
Networking	OutNet.	a: 10GB b: 15'000GB c: 200'000GB
Object Storage	OS	a: [10GB storage - 10'000 requests type A - 10'000 requests type B] b: [1'000GB storage - 0 requests type A - 50'000 requests type B] c: [0GB storage - 20'000 requests type A - 0 requests type B] d: [0GB storage - 0 requests type A - 20'000 requests type B]
Disk	HD	a: 20GB b: 200GB c: 2'000GB

TABLE IV
AWS RESULTS

Name	COCA results(USD)	AWS calculator results (USD)
I1 (t2.micro)	a: 0.012 b: 4.392 c: 8.784	a: 0.012 b: 4.40 c: 8.79
I2 (m4.xlarge)	a: 0.215 b: 78.69 c: 157.38	a: 0.215 b: 78.69 c: 157.38
I3 (r3.large)	a: 0.166 b: 60.756 c: 121.512	a: 0.166 b: 60.76 c: 121.52
OutNet.	a: 0.81 b: 1326.11 c: 13891.11	a: 0.81 b: 1326.11 c: 13891.11
OS (S3)	a: 0.284 b: 23.02 c: 0.1 d: 0.008	a: 0.29 b: 23.02 c: 0.1 d: 0.01
HD (EBS)	a: 2.0 b: 20.0 c: 200.0	a: 2.0 b: 20.0 c: 200.0

A. Comparing COCA-PT with cloud providers pricing calculators

The services and contexts used to compare COCA-PT and cloud providers pricing calculators (here Google and Amazon) are presented in table III. Four contexts are considered: *a*, *b*, *c* and *d*.

Table IV (resp. Table V) presents the results of the contexts for AWS (resp. Google Cloud). The effective differences are in fact due to the rounding method used by the calculator. The AWS calculator uses a 2 decimals superior rounding while Google uses a 2 decimal nearest rounding. Applying the same rounding method to COCA-PT would result in the same results.

B. Comparing COCA-PT with Cloud providers bills

Verifying using usage bills is the best way to validate the COCA-PT tool. In fact, most of the providers don't have a pricing calculator. It is also very important to use collected real world data to prove the correctness of the COCA-PT system. We used two cloud applications to test COCA-PT: smarthepia and iCeBOUND.

TABLE V
GOOGLE CLOUD RESULTS

Name	COCA results (USD)	Google calculator results (USD)
I1 (n1-standard-1)	a: 0.05 b: 16.425 c: 25.55	a: 0.05 b: 16.43 c: 25.55
I2 (n1-highmem-16)	a: 1.008 b: 331.128 c: 515.088	a: 1.01 b: 331.13 c: 515.09
I3 (n1-highcpu-64)	a: 2.432 b: 798.912 c: 1242.752	a: 2.43 b: 798.91 c: 1242.75
OutNet.	a: 1.20 b: 1517.44 c: 16317.44	a: 1.20 b: 1517.44 c: 16317.44
OS (Google Cloud Storage)	a: 0.314 b: 26.02 c: 0.1 d: 0.008	a: 0.31 b: 26.02 c: 0.1 d: 0.01
HD (standard disks)	a: 0.8 b: 8.0 c: 80.0	a: 0.8 b: 8.0 c: 80.0

TABLE VI
SMARTHEPIA ON EXOSCALE

Resource - 5817min usage	Exoscale bill (USD)	COCA (USD)
1x Tiny (client)	1.35	1.3483
2x Small (database/API Server)	4.31	4.3068
3x Disks - 10GB	0.41	0.4074
OutNet. - 1.3GB	0.00	0.00
Total	6.063	6.0625

1) *smarthepia*: smarthepia [28] is an IoT platform that collects data from multiple sensors (luminance, motion, temperature, humidity) and controls connected switches (lights, blinds, radiators). It is deployed on the cloud using three instances. The first instance, called *client*, is responsible of collecting data from the sensors and storing it into another instance (database). The third instance is an API server which exposes database data and switches control. smarthepia is a service-oriented application. To run the tests, we deployed the application on Exoscale and AWS during a 4 day and about 57 minutes period. API calls were simulated using Apache JMeter, they are responsible of the network consumption. Table VI (*resp.* Table VII) shows the actual costs invoiced by exoscale (*resp.* AWS Amazon) and the costs calculated by COCA-PT.

2) *iCeBOUND*: The Cloud Based Decision Support System for Urban Numeric Data (iCeBOUND) [29] is used to calculate the solar energy potential of surfaces in urban landscapes. The system is assumed to identify good candidate roofs for installing solar panels. The goal is to quantify the solar power a roof could generate by taking into account local weather, roofs orientation and how much shade falls on it from nearby trees and buildings. iCeBOUND is a task-oriented application based on an irregular and time consuming shading algorithm[29].

To run the tests, we computed the solar potential map of an area of 3.4 km * 3.4 km, using a cluster of 6 instances. The calculation of the solar potential of this area took 232 minutes. Table VIII shows the invoice amounts generated by AWS Amazon and the costs calculated by COCA-PT. Rounding apart, the only difference is the InnerNet value. This value is the number of MB exchanged by the instances during runtime using their public network interface. This kind of networking

TABLE VII
SMARTHEPIA ON AWS

Resource - 97hours usage	AWS bill (USD)	COCA (USD)
1x t2.micro (client)	1.16	1.164
2x t2.small (database/API server)	4.46	4.462
3x Disks (EBS) - 8GB	0.32	0.32333
OutNet. - 1.3GB	0.09	0.09
Total	6.04	6.039333

TABLE VIII
ICEBOUND ON AWS

Resource - 232min usage	AWS bill (USD)	COCA (USD)
6x m3.2xlarge (1x master/5x slave)	12.77	12.768
6x Disks (EBS) - 8GB	0.03	0.02666
OutNet. - 913MB	0.00	0.00
InnerNet. using pub. IP - 600MB	0.01	N/A
Total	12.80	12.79466

activity is not yet supported by COCA-PT.

VI. CONCLUSION

This report presented a cost-optimized cloud application placement tool (COCA-PT) based on a resource consumption model (RCM). COCA-PT retrieves pricing data and metrics used to create the RCM of any cloud application. It also compute the deployment and execution of the application on cloud infrastructures targeted by the user. The “context” of the application (the “conditions” under which the application is deployed and executed) is supplied either manually by a user (usually through application profiling) or automatically by collecting live data from a multi-cloud Application Management Platform on which the application is deployed.

COCA-PT was evaluated in the concrete case of a multi-cloud Application Management Platform called SlipStream, using a variety of typical cloud applications. Differences between the actual or simulated costs and the cost calculated by our pricing module were 2% or lower, showing that the methods are accurate despite the limitations described in the text. An extensive study of concrete use cases cost optimization will be conducted once the tool is mature enough to evaluate its performance.

A. Limitations

The main problems underlined by this work are related to data collection for the pricing and monitoring. The absence of pricing and monitoring APIs influenced some choices for the implementation. The developed workarounds are functional but lack robustness. Hopefully, price list APIs will be more generally available in the future to allow a robust implementation for this aspect of the problem. However, monitoring of higher level services remains a significant issue. Without accurate monitoring information, creating a precise, complete resource consumption model for high level services (mainly SaaS/PaaS services) is impossible.

B. Perspectives

Despite the identified limits, much can still be done to improve and extend the pricing calculations. For cost optimization, including spot instances [19], preemptible instances [20], and reserved instances could provide better guidance on the optimal cloud provisioning, especially for service-oriented cloud applications. Other agents, in addition to the network monitoring agent, could be added to determine if the host VM resources are under-exploited. This would enable (automated) vertical scaling that could lead to cost reductions.

To further expand the system, the out-of-scope criteria like SLAs, availability of services, data locality constraints, and security constraints [24] could be taken into account. Including also cloud performance [3] [4] would also be a great addition to the system. Costs, performance, security, and level of service are all elements needed by a user for proper selection of a cloud provider.

We have only considered entire applications running in a single cloud. SlipStream already supports hybrid clouds and multi-cloud deployments. The system could be generalized to hybrid clouds, determining which part of a cloud application should be run where, potentially increasing performance and lowering costs.

VII. ACKNOWLEDGEMENTS

This work is part of the PAChA research project which is financed by the Swiss Commission for Technology and Innovation (CTI).

REFERENCES

- [1] Gabriella Laatikainen, Arto Ojala and Oleksiy Mazhelis *Cloud Services Pricing Models* From Physical Products to Software Services and Solutions: 4th International Conference, ICSOB 2013, Potsdam, Germany, 2013
- [2] B. Martens, M. Walterbusch and F. Teuteberg *Costing of Cloud Computing Services: A Total Cost of Ownership Approach* 45th Hawaii International Conference on System Sciences, 2012
- [3] Ang Li, Xiaowei Yang, Srikanth Kandula and Ming Zhang *CloudCmp: Comparing Public Cloud Providers* Duke University and Microsoft Research, 2010
- [4] Nane Kratzke and Peter-Christian Quint, *About Automatic Benchmarking of IaaS Cloud Service Providers for a World of Container Clusters* Luebeck University of Applied Sciences, Germany, 2015

- [5] Saurabh Kumar Garg, Steve Versteeg and Rajkumar Buyya *SMICloud: A Framework for Comparing and Ranking Cloud Services* Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference, p 210-218. IEEE, 2011
- [6] Hong-Linh Truong and Schahram Dustdar *Composable cost estimation and monitoring for computational applications in cloud computing environments* International Conference on Computational Science, ICCS 2010
- [7] Sivadon Chaisir, Bu-Sung Lee and Dusit Niyato. *Optimization of Resource Provisioning Cost in Cloud Computing* Services Computing, IEEE Transactions on, 5(2), 164-177. IEEE, 2012.
- [8] Amazon Web Services. Price list API. <https://aws.amazon.com/fr/blogs/aws/new-aws-price-list-api/>. Accessed May 10, 2016
- [9] Google Compute Engine. Price list API. <https://cloudpricingcalculator.appspot.com/static/data/pricelist.json>. Accessed May 10, 2016
- [10] Louis Columbus, Forbes tech. *Roundup Of Cloud Computing Forecasts And Market Estimates, 2016* <http://www.forbes.com/sites/louiscolombus/2016/03/13/roundup-of-cloud-computing-forecasts-and-market-estimates-2016/>. Accessed May 5, 2016
- [11] SixSq. SlipStream application management platform. <http://sixsq.com/products/slipstream/>. Accessed 2015-2016
- [12] Amazon Web Services. AWS EC2 pricing. <https://aws.amazon.com/ec2/pricing/>. Accessed 2015-2016
- [13] Google Compute Engine. GCE pricing. <https://cloud.google.com/compute/pricing/>. Accessed 2015-2016
- [14] Exoscale. Exoscale cloud pricing. <https://www.exoscale.ch/pricing/>. Accessed 2015-2016
- [15] Microsoft Azure. Azure cloud pricing. <https://azure.microsoft.com/en-us/pricing/>. Accessed 2015-2016
- [16] Ultimum. Ultimum cloud pricing. <http://ultimum.io/solutions/ultimum-cloud/>. Accessed 2015-2016
- [17] Rackspace. Rackspace cloud pricing. <https://www.rackspace.com/cloud/public-pricing/>. Accessed 2015-2016
- [18] IBM Softlayer. Softlayer cloud pricing. <http://www.softlayer.com/virtual-servers>. Accessed 2015-2016
- [19] Amazon Web Services. EC2 Spot Instances. <https://aws.amazon.com/ec2/spot/>. Accessed May 27, 2016
- [20] Google Compute Engine. Preemptible VM instances. <https://cloud.google.com/compute/docs/instances/preemptible/>. Accessed May 27, 2016
- [21] ISO. ISO 4217 currency codes. http://www.iso.org/iso/home/standards/currency_codes.htm. Accessed November, 2015.
- [22] UNECE. UN/CEFACT 3 letters code. http://www.unece.org/fileadmin/DAM/cefact/recommendations/rec20/rec20_rev3_Annex3e.pdf. Accessed November, 2015.
- [23] Schema.org. Schemas for structured data. <https://schema.org>. Accessed August-November, 2015.
- [24] Ferhat Guezlane. *CloudSF: A decision support system for the evaluation and selection of Cloud services adapted for the Swiss context* Internship thesis, HEC Lausanne, 2014.
- [25] Amazon Web Services. Simple Monthly Calculator <https://calculator.s3.amazonaws.com/index.html>.
- [26] Google Cloud Platform. Pricing Calculator <https://cloud.google.com/products/calculator/>.
- [27] Microsoft Azure. Pricing Calculator <https://azure.microsoft.com/en-us/pricing/calculator/>.
- [28] Smarthepia presentation page <http://lds.hesge.ch/smarthepia>.
- [29] Anthony Boulmier, John White and Nabil Abdennadher. *Towards a Cloud Based Decision Support System for Solar Map Generation* Cloud Computing Technology and Science (CloudCom), IEEE, December 2016.