OVERVIEW OF CURRENT COMMERCIAL PAAS PLATFORMS

Andres García-García, Carlos de Alfonso and Vicente Hernández
Instituto I3M, Universitat Politècnica de València, Camino de Vera s/n, Valencia, Spain

Keywords: Cloud computing, PaaS platform, Platform-as-a-service, Web services.

Abstract: The development of Cloud Computing has paid attention mainly to the virtualization of hardware components, under a scheme of Infrastructure-as-a-Service (IaaS). Most of Cloud vendors offer solutions based in IaaS to offer virtual services and applications, instead of developing a proper Platform-as-a-Service (PaaS) solution. This paper analyzes some existing commercial PaaS clouds, identifying in what degree they meet some essential requirements. These requirements are also argued, and guidelines for addressing them using an academic approach in a PaaS solution are provided.

1 INTRODUCTION

Cloud Computing technologies have been traditionally associated with business environments. Therefore they have been used and promoted by enterprises to build their business models around that technology and its applications. This situation has caused enterprises to be the ones leading the development of this technology, putting commercial interests before scientific or academic interests and making researchers to follow trends leaving some problems inherent to the technology unattended.

Enterprises who have built their business model around Cloud Computing try to offer competitive solutions as soon as possible. Many of these solutions are not correct or formal solutions, and would require a deeper and elaborated research process, typical from academic environments. The need of having a product in the market has made commercial solutions to ignore some features which may have been considered useful when applied to this technology.

One of the ignored developments has been the Platform-as-a-Service (PaaS) level of Cloud Computing (Vaquero et al., 2009). Most of Cloud vendors just offer solutions based in the Infrastructure-as-a-Service (IaaS) level, providing virtual hardware resources in the form of virtual machines. This same concept has been used to offer virtual services, including preconfigured services, packaged in a virtual machine image which are deployed into IaaS solutions. This approach is easy and immediate to implement since it uses existing Cloud infrastructures, but it also encompasses some drawbacks as the waste of resources or a loss of control over direct services management.

In this context, it is interesting to follow an approach to PaaS level based in building a platform from the scratch with the aim of addressing service virtualization directly, not by means of machine virtualization. In order to build this platform, a set of features are needed to be defined, to build a correct, reliable and robust system.

This paper performs a comparative analysis between currently available commercial PaaS solutions. After this comparative, some features that these solutions do not address are stated, detailing for each one how the platforms implement them, and why are considered incorrect. Also some guidelines to open research lines in these features are offered, providing researchers with a broad specter of fields to provide further developments.

2 PaaS PLATFORMS ANALYSIS

In order to detect open problems and unattended features in commercial PaaS Cloud solutions, a comparison has been made between four currently available commercial PaaS platforms: Windows Azure (Microsoft Corporation, 2010c), Google App Engine (Google Inc., 2010a), Heroku (Heroku Inc., 2010) and Engine Yard (Engine Yard, Inc., 2010). These platforms are well known examples of commercial PaaS Clouds, and the most representative part of the state-of-the-art of PaaS platforms. All four solutions have a clear web application orientation, although each one

García-García A., de Alfonso C. and Hernández V..
OVERVIEW OF CURRENT COMMERCIAL PAAS PLATFORMS.
DOI: 10.5220/0003604703680376
In *Proceedings of the 6th International Conference on Software and Database Technologies* (IWCCTA-2011), pages 368-376
ISBN: 978-989-8425-76-8

takes a different approach.

- **Windows Azure** is Microsoft's major product for the Cloud Computing market. It offers a general computing runtime for .NET, PHP, Ruby, Java and Python languages, achieving the highest quantity of supported languages from all four solutions. Azure offers an SDK integrated with Visual Studio for .NET and SDK and Eclipse plug-ins for other languages, assisting developers with application templates and auto deployment capabilities.

  The platform hosts two application roles, Web and Worker, where Web role acts as the user interface, and as the front end for the applications, while Worker role does background calculations. Both roles communicate by exchanging messages, using a built-in message queue. Applications can use several interfaces such as databases, blob storage, authentication, etc. for extra functionality.

  Azure uses two billing approaches, either a monthly payment or a pay-as-you-go schema. Using monthly payment, developers purchase a quantity of resources to use along a month for a fixed price. In the pay-as-you-go schema, billing is done monthly based in the amount of resources used in that time frame.

- **Google App Engine** is Google's PaaS solution. It supports Java and Python languages, and runs applications in a "safe sandboxed environment". Runtimes for both languages have been modified to impose restrictions over the capabilities of hosted applications. Specifically, this sandbox environment prevents applications from:

  – Writing to the filesystem.
  – Opening a socket or access another host directly.
  – Spawning a sub-process or thread.
  – Making other kinds of system calls.
  – Only classes in a whitelist can be accessed from the Java standard library.

  These limitations are oriented towards guaranteeing the correct behavior of applications and the safety of the system. There exist additional limitations, such as the 30 second limit for an application to send a response from a request, that are intended to improve the platform performance.

  Google App Engine includes an extensive monitoring system which provides detailed statistics and metrics about the applications performance, including resources usage (CPU, bandwidth, storage), number of requests received, number of API calls, etc. These values, or quotas, are used as a measure for the payment for the service. Service hosting is free under some values for each quota, but it is set unavailable when these quotas are exceeded. If billing is enabled, it switches to a pay-as-you-go model for quotas over the free limit. Developers are able to specify a budget which limits the quantity of resources to hire from the platform. Each quota resets its value in a 24 hours period, and has a max value that cannot be overcome even using the payment model.

- **Heroku** is a company founded in 2007 which focuses in offering a PaaS platform for Ruby applications. Developer's applications are hosted in a software stack (named Dyno) which include frameworks (Rails (Hansson, 2010b), Sinatra (Mizerany, 2010)), middlewares (Rack (Neukirchen, 2010), Rails Metal (Hansson, 2010a)) and a VM (MRI (Matsumoto, 2010)) for the Ruby language, wrapped in virtual machine images that are deployed in an IaaS infrastructure (specifically in Amazon EC2 (Amazon.com Inc., 2010)). These Dynos are organized in a pool of applications which receive requests from users. A reverse proxy and a HTTP cache manage external requests serving applications via a routing mesh, which acts as a load balancer between dynos. Storage (either persistent or not) is provided by means of a database system or a memory cache system.

  Heroku offers two roles: dynos (equivalent to a web role in Azure) and workers. Dynos process HTTP requests from users, while workers perform background tasks and transmit messages via message queue. Developers can choose dynamically the number of dynos and workers that they want to hire, with a minimum of 1 dyno and a maximum of 24 for both. Databases plans offer access to a databases system with different capacity or performance level. The user is able to choose between shared or dedicated databases and the number of cpu units.

  Finally, an add-on system provides extra functionalities to hosted applications, for instance, capacity to use SSL, a cron process, custom domains, etc. Databases and add-ons are provided in a monthly fee basis, while dynos and workers are billed hourly, based in the number of units hired.

- **Engine Yard** is a company founded in 2006 which provides a Ruby Cloud, using an underlying IaaS infrastructure (Amazon EC2) very similar to Heroku. Engine Yard however focuses its business model around the technical support and consultative services, instead of application hosting.

Engine Yard cloud is built over Amazon EC2 and Amazon S3, and it provide wrapped ruby environments as virtual machines. Customers are able to purchase different sizes of these instances and add additional features like a dedicated database or a load balancing cluster. Once some instances have been purchased, developers can deploy and manage their applications from a web interface within Engine Yard website. That interface acts as a front end for the platform and includes different managing operations. Billing is based in the instances hired, the number of hours running, storage and data transfer.

Recently Engine Yard has divided their Cloud offering in two branches, one named AppCloud, which offers the described standard features, and another "Enterprise Grade Cloud" named xCloud. xCloud offers a set of extra features not available in public AppClouds, like the provision of an auditable infrastructure. The solution offers plans such as a fractional cluster offer, which includes dedicated database and disk storage, or a dedicated cluster, an exclusive solution where the infrastructure is reserved just for one customer. Additional features include consultative services, oriented to help enterprises to migrate their applications to the cloud, providing extensive analysis about the application needs, and a customized solution in the Engine Yard platform, while a premium support system extends standard support to 24x7 availability for critical issues, proactive monitoring of applications, etc.

The following features have been analyzed for these platforms.

- Underlying architecture: This feature specifies which software architecture supports the solution.
- Status: This topic states whether the solution is available for the users for creating private clouds or it is proprietary.
- Use of standards: It is also analyzed whether the solution makes use of standards for three main operations: authentication mechanisms to access the platform, access to databases for built-in storage support and communication protocols to interact with the platform.
- Interoperability: States whether the platform is able to interoperate with other clouds or it is isolated
- Versioning: Refers to the support of the platform to software versioning.
- Migration: This topic states whether the solution supports service migration.
- Catalog: It refers to whether the platform provides some sort of information services, that would include any kind of useful information about the platform or the services.
- SLA: This feature is related to the mechanisms included in the platform for defining the obligations, warranties, compensations or pricing of the resources in the terms of the service between provider and customers. These mechanisms may be offered in several options and features:
  - Text document SLA: The provider uses a legally binding document which acts as a contract between the company and the customers.
  - Purchasing options: The user is able to purchase different business plans which offer specific functionalities for a fee (e.g. fixed size monthly plans, additional support, etc.).
  - Configuration options: Users can specify restrictions over their services using configuration options (e.g. geographical deployment constraints).
  - Pay-as-you-go: Payment is decided dynamically over the quantity of resources used and the options that have been purchased.

The comparison shown in Table 1 highlights that there are some topics which are not supported by the platforms analyzed. In the following section, based in the data gathered from the analysis of the commercial platform, a set of open problems in PaaS Cloud solutions is exposed.

3 OPEN PROBLEMS IN PaaS CLOUD PLATFORMS

This section reviews features that are interesting to be provided by PaaS platforms, and are widely provided by the current available PaaS solutions analyzed. The proposed features are the result of the comparison with the characteristics of Cloud concepts introduced by the generalization of IaaS platforms (Armbrust et al., 2009), and others identified by the community. For each of the topics an explanation of the approach of the commercial platforms is provided, and the explanation of its inadequacy from an academic point of view. Some guidelines to address the problems derived by that approach are also provided.

3.1 Pure PaaS Architecture

In general terms, details about the underlying architecture of the commercial PaaS solutions are not

Table 1: Comparison of commercial PaaS platforms.

Name	Windows Azure	Google App Engine	Heroku	Engine Yard
Underlying architecture	Not disclosed	Not disclosed	IaaS	IaaS
Status	Proprietary	Proprietary	Proprietary	Proprietary
Use of standards	SQL language for databases.	SQL language for databases.[1]	Keypair authentication. SQL language for databases.	Keypair authentication. SQL language for databases.
Interoperability	No	No	No	No
Versioning	No	Yes	No	No
Migration	No	No	No	No
Catalog	Service marketplace[2] "Codename Dallas".	Service marketplace[2] "Google App Marketplace". Dashboard with service statistics.	No	Dashboard with service statistics.
SLA	Text document SLA. Purchasing options. Pay-as-you-go. Configuration options.	Text document SLA.[1] Pay-as-you-go.	Purchasing options.	Text document SLA. Purchasing options. Pay-as-you-go.

known, but from what is shown, in most cases there does not exists any specific PaaS oriented architecture which supports each solution. The major difference between IaaS solutions and PaaS solutions is that IaaS virtualizes machines while PaaS virtualizes services.

The architecture of some platforms, such as Heroku, are based in an underlying IaaS solution without any specific consideration about services virtualization. The approach of building a PaaS solution over an IaaS solution is easy and fast, because both share many features as a starting point. According to that premise, the PaaS platform may be confident in the fact that the IaaS platform will provide some of the "cloud" characteristics (e.g. load balancing, isolation, migration, multitenancy, etc.).

However, while machine virtualization fits perfectly in an IaaS environment, it misses the aim of service virtualization. In a PaaS solution, the virtualized resource which is offered to clients is the runtime where services run, but some solutions (Heroku, Engine Yard) wrap this runtime inside a virtual machine image, instead of addressing virtual services directly. A PaaS architecture built over an IaaS architecture will take profit from the solutions to the problems found for IaaS architectures, but it should still address PaaS specific problems. As a result, the solution will not be able to solve some problems related with a PaaS platform or will do so in a less efficient manner, due to a biased vision of the PaaS solution.

This approach implies a high overload and a less direct management of the services. On one side, if a whole virtual machine is used, instead of just a virtual service, extra memory, storage and processor requirements are added to those that the service needs. Moreover the management of the services becomes more complicated, as it is necessary to work at machine level instead of service level.

Other platforms, like Google App Engine, seem to be based in an original PaaS architecture, although no details are revealed about it. This approach introduces new considerations about the management of services, like imposing limitations in the service behavior, but also provides several benefits like a versioning system or detailed statistics reports.

While the typical approach is to embed the runtime in a virtual machine and use it as a building block, the proposed approach is to virtualize the runtime itself, and use it as a building block. Using runtime virtualization (or virtual containers) researchers are able to design a pure PaaS architecture. Also this approach bring new needs, problems and features that must be addressed, which are missed when virtual machines and IaaS solutions are used as a base. Service virtualization offer new models for monitoring and billing, SLA terms, interoperability features, security concerns, etc. Furthermore, runtime virtualization increases the flexibility of the system, since virtualized runtimes can be placed into virtual machines, and a PaaS solution can be built using either IaaS solutions, physical machines directly, or even applying mixed schemes.

New research lines opened by this feature include defining the requirements of a PaaS platform and the specification and design of a PaaS oriented architecture.

[1] Introduced in App Engine for Business beta, in May 2010.

[2] Service Marketplace are web portals where developers and business publish they applications and users buy subscriptions for them.

3.2 Tools for the Development of Private PaaS Clouds

In order to use an IaaS cloud, users can adopt solution offered by IaaS vendors or build a private cloud using their own resources and a cloud software (Nurmi et al., 2008)(Sotomayor et al., 2009). When considering PaaS level, there are few vendors who offer private PaaS solutions, and even less tools that enable users to build their own private PaaS cloud, like (University of California, Santa Barbara, 2010).

Using a cloud from a vendor means the outsourcing of management and maintenance operations and leverages the transition between traditional datacenters and cloud computing for enterprises. This is often considered as a desirable feature since developers just use the resources without additional management cost such as physical machine hosting, cooling systems, network access, etc. However, sometimes private cloud solutions meet the requirements of users better than public clouds.

One of the major concerns about outsourcing hosting services is the security and privacy requirements. Enterprises have the responsibility of hosting services that have a high value (known as mission critical services) and storing information which contains sensible data (e.g. clients personal information) in order to keep their compromise with customers and maintain their business model. When services are outsourced to external companies, enterprises require from vendors guarantees that are compatible with those that they offer to their customers. Also outsourcing implies a loss of control over services and data, which force enterprises to rely in third party hardware and staff to run and manage their assets.

When building a private PaaS cloud, developers are able to host mission critical services and sensible data in house, avoiding security concerns and keeping direct control over software and resources. Also, the user takes profit from all the other benefits provided by cloud computing such as resource optimization or easier management.

Furthermore, hybrid clouds solutions may be built, where private clouds interact with public clouds. In such solution, private clouds manage sensitive data and services, while public clouds provide extra capacity to the system.

Under the scope of IaaS, there are several initiatives such as Eucalyptus or others, which are derived from the usage and the concepts introduced by Amazon WS. Unfortunately, there are few PaaS initiatives, and that is probably why most of the concepts are still under development.

Tools for the development and deployment of PaaS solutions would allow users to build their own private clouds and taking advantage of its benefits. Using an open and free tool, administrators are able to deploy PaaS solutions, configuring and adapting the software to their needs. Therefore, the design and implementation of an open tool for the deployment of PaaS infrastructure remains as an open research line itself.

3.3 Use of Standards

As it occurs with any emerging technology, Cloud Computing yet lacks its own solid standards for many operations. However, that void has become a chance for some companies to use technology of their own interest in parts where some standards could have been applied (e.g. LiveID (Microsoft Corporation, 2010b) authentication of Windows Azure, Google App Engine using GQL (Google Inc., 2010c) for databases).

Cloud Computing has been defined as the new model for Utility Computing. One of its most important features is that anyone can plug their systems to “the cloud” and use computing power, storage or network as a commodity. However, if standards are not used, the purpose of a true Utility Computing is lost, since the developed systems cannot be used in any but just their target cloud.

The lack of standards produces vendor lock-in and it also difficult interoperability between clouds. While the definition of standards for the cloud as a whole is a work in progress (Open Cloud Consortium (OCC), 2010), the usage of standards for key operations (communications, authentication) should be encouraged. A system with a highly modular design would provide a good basis to adopt standards and would enable the upgrade of them to new versions (or new standards).

An open research line in this field consists in identifying key components where standards can be used, and design protocols and procedures which are part of the state of the art specifications for those operations, helping the joint effort for the definition of cloud standards.

3.4 Platform Utilities

The paradigm shift between IaaS and PaaS consist on the provisioning of a platform for the deployment and management of services, instead of the provisioning of virtualized hardware. However a platform not only encompasses the deployment and execution of software, but to give support to the whole development cycle, from implementation to debugging.

A complete PaaS solution should provide a set of Platform Utilities that effectively realize a higher abstraction level than the IaaS solutions. A PaaS product that only stores and executes services does not provide many advantages over an IaaS solution that provides on-premise virtual machines configured to execute services.

The generic term Platform Utilities includes a set of features that the platform offers to developers and hosted services in order to integrate the Cloud deployment in the software development cycle. Some of these utilities are described below.

- IDE integration. One of the aims of PaaS solutions is to integrate the cloud deployment with the software developing process. The underlying idea is that the whole development procedure should be cloud-aware, to enable developers to create cloud applications and not applications that run in the cloud, adapting the development cycle to the cloud. For achieving this integration is necessary to provide an IDE that communicates with the Cloud and allow testing applications before deployment, automatically deploying applications in the Cloud and using other Platform Utilities. In this field, Microsoft offers Azure Tools for Visual Studio, an add-on to the Visual Studio IDE which allows local testing of Cloud services in a simulation environment, the usage of the Azure libraries and automatic deployment of applications in the Azure Cloud. Google offers the Google App Engine SDK (for Python and for Java), as a set of tools with similar functionalities, and the Google Plugin for Eclipse, which allows the usage of the SDK in an IDE.
- Communication System. As in any distributed system, applications may have the need to communicate between them. Some software solutions, such as a master-slave execution schema, make heavy use of inter-service communication, and therefore a built-in efficient solution leverages the load of the system and improves the development cycle. In this field, Microsoft Azure includes the Service Bus utility, which allows clients and services to exchange messages or data.
- State Replication System. One of the principles of the Cloud is the scalability of services by means of multiple services instances, load balancing and stateless interactions. The principle of stateless interactions says that in order to properly serve a petition, a service must not rely on previous interactions, and therefore the same petition may be served by different instances, producing the same return value. However many applications need the management of some kind of state, that must be consistent between instances of a service. This requirement is dealt with the usage of Cloud Storage, which typically incurs in computation overhead and economic cost. The usage of state replicating mechanisms would allow developers to keep the state of their services consistent using built-in platform mechanisms and without the need of Cloud Storage.
- Versioning. This feature allows developers to maintain an incremental development cycle, and enables testing and upgrading applications. Developers are able to keep long-term services and add new functionalities incrementally, and testing the new software versions in the target environment before they are released. Developers would be able to deploy stable versions of software in the cloud, and concurrently deploy beta versions for testing purposes. Once beta versions have been debugged and became stable, the deployed instances would be able to be update on the fly, providing users with stable and updated services transparently. Most of current PaaS solutions don not take into account the software versions, and developers must manually manage those concepts in a non-transparent manner. Google App Engine is the only analyzed platform which integrates the concept of versioning, allowing developers to define service versions, deploy several at the same time and define one of them as default, which will be the one publicly available.
- Debugging Mechanisms. Debugging of distributed software is a challenging issue and a field of study itself, and it becomes even more complex when the developers are not the owners, nor have direct access to the platform executing the code. Local testing of services may solve some errors before the service becomes publicly available, but only when the software to test becomes used in a real world scenario, all possible issues become noticeable. Since developers do not have direct access to the virtual runtimes, the platform itself must provide mechanisms to debug deployed services. In this field, Google App Engine provides a logging service that allows developers to log error or warning systems, and these messages are retrieved by the platform and displayed in the dashboard, where developers can check them. Heroku offers the Exceptional add-on, with two subscription levels (regular and premium), a system which stores and make real-time notification to developers when an exception arise in a running application.

The inclusion of Platform Utilities provides a clear differentiation in terms of abstraction level be-

tween the IaaS and PaaS solutions. A PaaS solution that provides Platform Utilities allows building Cloud-aware applications, that actively make use of its features for their advantage.

The cited Platform Utilities, among other that may be formulated for different use cases, represent an open research line in this field.

3.5 Service Migration

Live migration is a concept associated to distributed systems, specifically SOA (Service Oriented Architecture), although the popularization of virtualization technologies has recently associated the term to virtual machine migration. Live migration is defined as the movement of a running piece of software from one physical host to another one without any disruption of service or perception of downtime from the users point of view (Clark et al., 2005). The major advantages of live migration are the following:

- Software can be moved from one machine to another, making its execution independent from the original machine. This ability allows administrators to interact with the original machine without affecting the application. For instance, if a machine needs to be shutdown for maintenance, live migration would enable services hosted in it to keep running in other host machine with no downtime associated.
- Services can be migrated between resources, applying load balancing algorithms. If a machine is overloaded and other has spare capacity, moving services between both of them would optimize resource usage and improve performance.
- Live migration is done without disrupting services. Users can keep using services without noticing that they have been moved to another physical machine
- Using interoperability and live migration, developers can change its cloud provider automatically and without disrupting its customers.
- Autoscaling (up and down) becomes easier and more efficient. When scaling up, load can be redistributed among new nodes using live migration, while when scaling down, services from machines that will be stopped can be moved to the remaining ones.

Among the platforms analyzed, only Heroku explicitly apply some sort of migration in order to load balance resources, while the others totally ignore this concept, or hides it from users. Heroku is built on top of an IaaS infrastructure and it migrates its components (dynos and workers) between nodes in order to improve performance transparently to users. The migration consists in moving the virtual machine which hosts the service from a node to another.

Live migration of running instances has been seen as a desirable feature also for IaaS solutions, since it provides load balancing and better management of resources. As well as IaaS platforms continue working on offering virtual machine live migration, PaaS platforms should offer virtual services live migration.

Live migration, which is a field of study itself, can be achieved using a migration protocol. Those protocols often include three basic steps: deploy a new instance, redirect clients from the old instance to the new one and undeploy the old instance. This protocol works properly when services are RESTful (Fielding, 2000), specifically if they are stateless. However, if services keep some kind of state (either as local storage or as in-memory data) this protocol must include a state transfer step, which increases the complexity of the process. Therefore live migration involving state transfer opens new lines of research along with the service migration protocols.

3.6 Digital SLA

In a cloud environment, SLA becomes the contract that specify the terms of the agreement between the vendor and the user at a software level, specifying obligations, warranties and compensations.

Nowadays cloud vendors offer a very basic vision of SLA for their solutions. They provide a plain text document that specifies obligations in terms of uptime, stating that vendors will offer x% uptime (measured along a certain amount of time), and the compensation in case of violation of the terms, commonly formulated as a discount in the next charge to the client. Other candidate options to be part of the SLA (geographic constraints, CPU, memory requirements) are left out as configuration options, or totally ignored in some cases.

In order to offer a complex SLA system which can handle service restrictions, configuration, user information, etc., platforms need to incorporate the concept of ‘Digital SLA’ and a SLA framework to allow administrators to define their own SLA terms and management rules. This concept has been included in the work lines of other projects such as Reservoir (Rochwerger et al., 2009) which defines the inclusion of SLA metadata in the manifest of the services, and is the key topic in the SLA@SOI project (The SLA@SOI consortium, 2010), whose aim is to provide a business-oriented Service Oriente Infrastructure (SOI) which allow the management of the business processes by means of Digital SLAs.

Digital SLAs would be the binary representation of the documents which define the terms of the service between provider and users. Those terms should include every element that defines the agreement between both parts:

- Resources required by the user (CPU, memory, storage).
- User restrictions over services (e. g. geographical constraints).
- Price of the elements, which could be specified individually for each instance or service.
- Configuration options over services at platform level (e. g. elasticity rules, list of developers allowed to manage a service, number of replicas for stored data).
- Warranties offered by the provider (maximum response time, security constraints, etc.).
- Compensation terms (compensation to users when SLA is violated). It would be interesting to specify them individually for each element, depending of the 'severity' of the issue.

This Digital SLA, unlike traditional SLA contracts expressed as plain text documents, can be stored and automatically managed by the platform, enabling to deal with it in an automated manner. A platform managing Digital SLA would be able to automatically perform operations to correct or prevent SLA violations, report the incident to administrators and developers, and take corrective actions and compensate users as required.

New open research lines derived from this field includes the definition of the terms of the SLA, definition of a language to represent SLA, creation of protocols for SLA manipulation, interaction of SLA with a PaaS platform components, definition of mechanisms to enforce SLA, prevent their violation, take corrective actions and compensation to users in case of violation, etc.

3.7 Information Services

Some of the features of PaaS Cloud platforms need a system to store and retrieve different kinds of information, such as monitoring information, metadata for services and users, etc. In a Cloud solution, information is generated everywhere in the system, either by the platform components or by external users interacting with them. Some of this information must be retrieved and stored for further processing (for instance the CPU load of a service for SLA monitoring), and therefore arise the need of such a system.

In the field of distributed systems, this concept is realized as Information Services, or Catalogs. A Catalog is a semi-centralized system which store and manage information, and provides an API for every component to add or retrieve data from it.

Current PaaS solutions have made a particular interpretation of the Catalog concept. Google and Microsoft offer Google Apps Marketplace (Google Inc., 2010b) and Microsoft codename Dallas (Microsoft Corporation, 2010a) services. Both products are web portals where developers can register their applications hosted in the cloud and offer them for a fee, and consumers can browse for and subscribe to them. These services offer a commercial trade point for developers and consumers, but do not offer the features of a Catalog.

Besides the App Marketplace, Google App Engine includes an integrated dashboard where developers can check statistics and metrics about their services, as well as an API to allow hosted services to check some monitoring information about themselves. This concept comes closer to an Information Service, as defined in (Czajkowski et al., 2001), but yet lacks the global nature of such system.

Open research lines in this field comprise defining an organization of a catalog for a Cloud solution, defining what information is included in it, how the information is stored and managed, define replication and distribution mechanisms, define the access protocols and API for the addition, modification, deletion and retrieval of information, etc.

4 CONCLUSIONS

The lack of research in the Platform-as-a-Service level of cloud computing have produced an absence of detailed analysis about its features, needs and challenges. The state of the art in PaaS clouds have been mainly carried out by enterprises, which have released PaaS solutions guided by commercial interests and not by scientific or academic considerations. Their main aim usually is to deliver solutions to their customers in a short time, in order to gain advantage over their competitors. This business model has caused the release of platforms constrained by the limitations of the model established by each provider.

Although the reviewed platforms conform a representative sample of the current state of the art in commercial PaaS Cloud, not every platform provides the same set of features. In order to provide efficient, reliable and robust products, a description of the basic characteristics needed to build a PaaS solution has been made, by means of an analysis of the state of the

art of Cloud Computing, but also incorporating part of the expertise of Grid Computing and IaaS solutions.

This paper highlighs each one of these selected features, and explains its motivation and importance. Also it defines the degree of compliance of each platform for each feature, and includes a brief description of the challenges associated with them.

Future works includes further research in PaaS solutions requirements in order to provide more features and refine the definition of the ones included in this work. For each one of these features identified, a set of open problems or research lines provide a broad range of new developments that not only concern PaaS Cloud, but all levels of Cloud Computing as a whole. Some of these research lines include but are not limited to the design of live migration procedures, specification of a SLAs, specification of a Cloud Catalog, etc.

ACKNOWLEDGEMENTS

The authors wish to thank the financial support received from the Spanish Ministry of Education and Science to develop the project “ngGrid - New Generation Components for the Efficient Exploitation of eScience Infrastructures”, with reference TIN2006-12890. This work has been partially supported by the Structural Funds of the European Regional Development Fund (ERDF). This work has been developed under the support of the program “Formación de Personal Investigador de Carácter Predoctoral” (grant number BFPI/2009/103), from the “Conselleria d‘Educació” of the “Generalitat Valenciana”.

REFERENCES

Amazon.com Inc. (2010). Amazon EC2 (Elastic Compute Cloud). http://aws.amazon.com/ec2/ [Online; accessed 01-03-2011].

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the Clouds: A Berkeley View of Cloud Computing. Technical report.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. In *NSDI’05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA. USENIX Association.

Czajkowski, K., Fitzgerald, S., Foster, I., and Kesselman, C. (2001). Grid information services for distributed resource sharing.

Engine Yard, Inc. (2010). Engine Yard Cloud. http://www.engineyard.com/ [Online; accessed 01-03-2011].

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis. Chair-Taylor, Richard N.

Google Inc. (2010a). Google App Engine. http://code.google.com/appengine/ [Online; accessed 01-03-2011].

Google Inc. (2010b). Google App Marketplace. http:// www.google.com/enterprise/marketplace/ [Online; accessed 01-03-2011].

Google Inc. (2010c). GQL. http://code.google.com/appengine/docs/python/datastore/gqlreference.html [Online; accessed 01-03-2011].

Hansson, D. H. (2010a). Introducing Rails Metal. http://weblog.rubyonrails.org/2008/12/17/introducing-rails-metal [Online; accessed 01-03-2011].

Hansson, D. H. (2010b). Ruby on Rails. http://rubyonrails.org/ [Online; accessed 01-03-2011].

Heroku Inc. (2010). Heroku. http://heroku.com/ [Online; accessed 01-03-2011].

Matsumoto, Y. (2010). Matz’s Ruby Interpreter. http://www.ruby-lang.org/ [Online; accessed 01-03-2011].

Microsoft Corporation (2010a). Codename “Dallas”. http://www.microsoft.com/windowsazure/dallas/ [Online; accessed 01-03-2011].

Microsoft Corporation (2010b). LiveID. https:// accountservices.passport.net/ [Online; accessed 01-03-2011].

Microsoft Corporation (2010c). Windows Azure. http:// www.microsoft.com/windowsazure/ [Online; accessed 01-03-2011].

Mizerany, B. (2010). Sinatra. http://www.sinatrarb.com/ [Online; accessed 01-03-2011].

Neukirchen, C. (2010). Rack: a Ruby Webserver Interface. http://rack.rubyforge.org/ [Online; accessed 01-03-2011].

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2008). The Eucalyptus Open-source Cloud-computing System. In *Proceedings of Cloud Computing and Its Applications*.

Open Cloud Consortium (OCC) (2010). Open Cloud Consortium (OCC). http://opencloudconsortium.org/ [Online; accessed 01-03-2011].

Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I. M., Montero, R., Wolfsthal, Y., Elmroth, E., Caceres, J., Ben-Yehuda, M., Emmerich, W., and Galan, F. (2009). The RESERVOIR Model and Architecture for Open Federated Cloud Computing. *IBM Journal of Research and Development*, 53(4).

Sotomayor, B., Montero, R. S., Llorente, I. M., and Foster, I. (2009). Capacity Leasing in Cloud Systems using the OpenNebula Engine. Cloud Computing and Applications 2008 (CCA08).

The SLA@SOI consortium (2010). SLA@SOI. http://sla-at-soi.eu/ [Online; accessed 01-03-2011].

University of California, Santa Barbara (2010). AppScale. http://appscale.cs.ucsb.edu/ [Online; accessed 01-03-2011].

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