Cloud Federation

Tobias Kurze*, Markus Klems†, David Bermbach†, Alexander Lenk‡, Stefan Tai† and Marcel Kunze*
*Steinbuch Centre for Computing (SCC)
Karlsruhe Institute of Technology (KIT), Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
Email: {kurze, marcel.kunze}@kit.edu
†Institute of Applied Informatics and Formal Description Methods (AIFB)
Karlsruhe Institute of Technology (KIT), Kaiserstrasse 12, 76131 Karlsruhe, Germany
Email: {markus.klems, david.bermbach, stefan.tai}@kit.edu
‡FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
Email: {lenk}@fzi.de

***Abstract*—This paper suggests a definition of the term *Cloud Federation*, a concept of service aggregation characterized by interoperability features, which addresses the economic problems of vendor lock-in and provider integration. Furthermore, it approaches challenges like performance and disaster-recovery through methods such as co-location and geographic distribution. The concept of Cloud Federation enables further reduction of costs due to partial outsourcing to more cost-efficient regions, may satisfy security requirements through techniques like fragmentation and provides new prospects in terms of legal aspects. Based on this concept, we discuss a reference architecture that enables new service models by horizontal and vertical integration. The definition along with the reference architecture serves as a common vocabulary for discussions and suggests a template for creating value-added software solutions.**

***Index Terms*—Cloud Computing, Cloud Federation, Reference Architecture, Lock-In, Hold-Up, Integration**

I. Introduction

The Cloud Computing paradigm advocates centralized control over resources in interconnected data centers under the administration of a single service provider. This approach offers economic benefits due to supply-side economies of scale, reduced variance of resource utilization by demand aggregation, as well as reduced information technology (IT) management cost per user due to multi-tenancy architecture [1].

These benefits have contributed to the increasing industry acceptance of Cloud services, which are seen as more affordable and reliable alternatives compared to traditional in-house IT systems and services. However, downsides of the Cloud Computing paradigm are surfacing. Surveys show that potential customers hesitate to outsource their business applications and data into the cloud [2]. Besides security concerns, application users are afraid of loosing ownership and control. The lack of standardized service interfaces, protocols and data formats is a portent of vendor lock-in [3]. This problem can lead to underinvestment, an economically inefficient situation, and therefore deserves our attention.

We propose an extended concept of *Cloud Federation* to enable the design of flexible and interoperable Cloud-based software, thereby lowering the adverse effects of vendor lock-in. We further discuss Cloud Federation as a key concept allowing the development of new types of applications.

The paper is structured as follows: Section II provides an overview of the state of the art on Cloud Stack and describes economic problems related to Cloud Computing. In Section III we state a definition of the term Cloud Federation and explain the concept in detail. Section IV introduces our vision of a reference architecture for federated Clouds. Finally, we give thought to open issues in Section V before concluding in Section VI.

II. Background and Related Work

Cloud Computing distinguishes the service models Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [4] [5]. IaaS offers infrastructure services, such as Compute Clouds, Cloud Storage, Message Queues, etc. PaaS offers complete platforms, solution stacks and execution environments, while SaaS is a software delivery model driven by a multi-tenancy architecture.

A. Cloud Stack

The principal service models IaaS, PaaS and SaaS do relate to one another and can be arranged as a stack. The IaaS layer represents the lowest level of the stack and is very close to the underlying hardware. Inside the IaaS layer two types of services can be differentiated: computational and storage [5]. Typical representatives for infrastructure services are Amazon's EC2 and Amazon's S3 (Appendix: Table A).

PaaS represents the second layer in the stack. Famous examples are Microsoft's Azure, Google's App Engine, SalesForce' Force.com and Amazon's Elastic Beanstalk (Appendix: Table A). Elastic Beanstalk is currently in beta phase and directly based on Amazon's IaaS offerings.

Upper layers such as SaaS (e.g., Google Docs) and Human as a Service (HuaaS) are directly or indirectly based on either IaaS or PaaS. Some secondary services, such as monitoring, accounting, authentication, metering or configuration and management are needed on multiple levels of the stack.

B. Cloud Software and Cloud Products

1) Private Cloud Computing Software: There is a broad spectrum of open source software, which mimics the proprietary systems of Amazon, Google, & Co. For example, Eucalyptus, AppScale, typhoonAE, and OpenNebula (Appendix: Table A). Users can install the open source software "in-house" as *private cloud* solutions. Since such a private cloud solution is partially compatible with the interfaces, protocols, programming models, and deployment options of the proprietary public clouds, this might be an approach to create an interoperable *hybrid cloud*, a composition of private and public clouds [6].

2) Cloud Marketplaces and Federation Offerings: While marketplaces, like Zimory or SpotCloud allow trading with Cloud resources, offerings like CloudKick and ScaleUp provide some federation functionality, e.g., monitoring and management supporting multiple clouds (Appendix: Table A).

C. Economic Theory

1) Vendor Lock-in: Vendor lock-in has been studied in economics research communities, for example by Robin Cowan [7]. Cowan identifies two sources of vendor lock-in: uncertainty of selecting an unknown technology, and the learning curve of a technology. The problem with two technologies A and B is formalized as the dynamic programming problem "Two-armed bandit".

We observe a growing number of Cloud Computing service providers and service offerings, in particular Cloud Storage and Compute services. These offerings tie users to a specific technology, which cannot be switched or replaced without significant switching cost. Apparently, this is the case for PaaS offerings, e.g., Google App Engine, which are closely integrated with proprietary services, such as Google user accounts and the Google e-mail service. Offerings like Amazon Web Services seem to have lower switching costs because they build upon Web service standards. However, a competing service provider would have to provide a similar technology (distributed system) with similar quality levels (availability, reliability, latency, throughput, etc.) and features (launch, stop, start, etc.).

This leads to the consequence, that users depend on the business strategy of the service provider.

2) Hold-up Problem and Underinvestment: The hold-up problem has been described by Klein, Crawford and Alchian [8] as being basically a contract problem. Two firms want to start business relations. In order to do so one party has to make an investment, which is specific in regard to the other party. Transferred to a concrete Cloud scenario, a company could invest in developers and applications, which are using Amazon's Web Services. This particular investment is of virtually no use when not used in the context of the two parties, i.e., the applications can not be used with Google's App Engine for example nor can the spezialized developers work with Microsoft's Azure. It is not possible to write complete contracts, i.e., contracts containing all, even future aspects of business relations, which might have an influence on the returns from the investment [9]. Due to incomplete contracts it is very likely that situations will arise that have not been foreseen at the time of the contract writing, making renegotiations necessary. In such future interactions one party may take advantage of the lock-in situation.

A party, anticipating the risk of a lock-in situation, typically takes suboptimal investment decisions, leading to underinvestment. [10, 11] When already facing the lock-in problem, a company may decide to stop further investment or to expend resources to protect itself against the lock-in. A party anticipating lock-in, hence, ends up in a hold-up situation, which in either case, leads to inefficient results [9].

Ewerhart et al. [12] summarized that in a lock-in situation, market forces are no longer effective and there is a risk of ex-post opportunistic behaviour. A party being forced to accept sub-optimal conditions cannot escape the situation due to the lock-in and finds itself in a hold-up [13].

III. Cloud Federation

Cloud federation comprises services from different providers aggregated in a single pool supporting three basic interoperability features - resource migration, resource redundancy and combination of complementary resources resp. services. Migration allows the relocation of resources, such as virtual machine images, data items, source code, etc. from one service domain to another domain. While redundancy allows concurrent usage of similar service features in different domains, combinations of complementary resources and services allows combining different types to aggregated services. Service disaggregation is closely linked to Cloud Federation as federation eases and advocates the modularization of services in order to provide a more efficient and flexible overall system.

We identify two basic dimensions of Cloud Federation: horizontal, and vertical. While horizontal federation takes place on one level of the Cloud Stack, e.g., the application stack, vertical federation spans multiple levels. In the following we focus on horizontal federation; aspects of vertical federation are out of the scope of this publication.

Several aspects of horizontal federation can be distinguished, e.g., provider domain and geography. Horizontal federation across provider domains may decrease provider dependency and thereby lower the risks of vendor lock-in and hold-up. Increased availability may be achieved through horizontal federation across multiple geographic regions. Also, vertical federation scenarios along similar aspects are imaginable.

Cloud Federation can be of interest for providers as well as for customers. Customers may profit from lower costs and better performance, while providers may offer more sophisticated services. However, hereinafter we focus on the customer perspective.

Two types of scenarios can be linked to Horizontal Federation:

- Redundancy: is used whenever there is a subset of (properly organized) service offerings that provide better utility to a client than any single service offering x_i, i.e., $\exists X \subseteq \bigcup_i x_i$ where $\forall x_i : u(X) > x_i$. the duration is, at least regarding a near time horizon, permanent as the user purposefully uses multiple service providers at the same time.
- Migration: can be triggered when a new service offering offers better utility to a client than any previously used service offering, i.e., $\exists x_{new} \forall x_i : u(x_{new}) > u(x_i) + u(c_s)$ where c_s are the total switching costs and $x_{new} \notin \bigcup_i x_i$.

Figure 1 illustrates the behavior over time of the two scenarios.

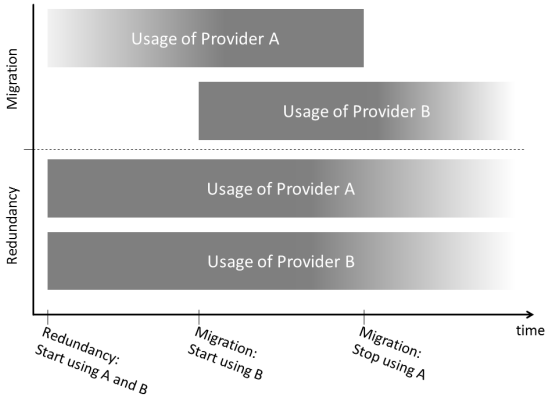

Fig. 1. Migration vs. Redundancy

A. Redundancy

Following the technical Cloud Stack [5], we can distinguish IaaS, PaaS and SaaS as different levels where horizontal redundancy can be used.

1) IaaS:

a) Compute services: know 3 kinds of redundancy:

- Redundant deployment: The same application logic is deployed to different providers. Still, incoming requests are processed by only one instance. Redundant deployment is used to increase the availability while decreasing provider dependence. Other reasons to do so could be compliance with regulations, which require instances in particular geographic locations. Also, customer proximity could be an issue to reduce latency.
- Redundant computation: The same application logic is deployed to different providers. Nevertheless, in contrast to redundant deployment here every request is processed by more than one instance. Reasons to do so could be either to improve performance by reducing the risk of an instance failing right before completing a task, an approach, e.g., taken in Google's MapReduce [14], or limited trust in the provider returning correct results.
- Parallel computation: Here, the data is broken down at bit level and processed at different providers' sites following the same application logic or complimentary services are deployed to different providers. Reasons for the 1st case could be security considerations where each provider only knows a tiny subset of the data. In the 2nd case, tasks are spread to the best fitting VMs to optimize latency and throughput.

b) Storage Services: know 3 kinds of redundancy:

- Replication: Data items are distributed as a whole and multiple copies are stored to increase availability while removing a single point of failure [15, 16, 17, 3] and reducing vendor lock-in. Furthermore, an increased number of replica may improve read latency due to customer proximity and increases durability. This is especially of interest when addressing resilience to correlated failures. However, whenever copies of the same data are kept at different sites there is a general tradeoff between consistency and availability as well as latency depending on how a storage system updates replica. This may happen synchronously, asynchronously in the background or as a combination of both.
- Erasure coding: Erasure coding uses RAID-like algorithms [18, 19] to distribute parts of data. If those parts overlap it is possible to restore data items even if a limited number of parts is missing. This obviously improves security as each provider knows only a tiny subset of the data item.
- Fragmentation: Here, items of type 1 are stored at provider A while type 2 is stored at provider B. This is useful when functional (e.g., data structure) and non-functional requirements (e.g., geographic location, durability, consistency) differ for different types of data.

2) PaaS:

PaaS offerings are hard to use redundantly as they usually not only follow a different programming model and support only a limited number of programming languages but also do applications developed for a particular PaaS offering make use of an entire ecosystem of services provided just within that PaaS offering. Furthermore, PaaS generally introduces limitations on the programming model they build upon so that applications need to be fine-tuned for a particular platform. So, the only sensitive alternative when trying to use federated PaaS offerings is to use one, for which an open source offering exists, which can, hence, be hosted by the customer or on top of IaaS compute resources. An example would be to redundantly use Google App Engine and AppScale running on top of Amazon EC2.

3) SaaS: Multiple SaaS offerings can be used redundantly with focus on different aspects:

- Focus on user experience: In this case, software services with similar functionality are used concurrently. An application could, e.g., allow the end user to toggle between visualization using Google or Bing Maps. This could enhance user experience by enabling a user to use a service he is used to. As a side effect, it would increase availability.
- Focus on availability: In this case software services with similar functionality are required but not used concurrently.

An application might switch over to a backup service in case of unavailability of the primary service.

While fine-grained SaaS offerings, e.g., Map services, can be used in a federation context relatively easy, it is very hard and probably cost-intensive to federate more complex services like, e.g., Salesforce. The difficulty to federate such offerings is caused by the fact, that it is virtually impossible to isolate smaller building blocks of the service as no competing solutions exist, which offer exactly the same functionality. Also, the potentially proprietary data formats and APIs of such services increase the problem. We believe that the issues related to the federation of SaaS offerings with larger granularity cannot be addressed in an adequate way by technical approaches and are therefore beyond the scope of this paper.

B. Migration

Migration incorporates scenarios where data respectively resources are being transferred from one Cloud provider A to another Cloud provider B. We identify two types of migration:

- Shadowed or redundant migration: In a migration scenario multiple similar services are usually only used for a limited amount of time, during which the old service is still operational while the new service is introduced. In the beginning, the new service is shadowing the old service to test it with live data. After switching over, the old service is shadowing the new one as a fallback solution in case of unanticipated failures. Finally, the old service is put out of service and the migration is finished.
- Non-redundant migration: Here, there is a hard switch-over. There is no shadowing period before or after.

In addition to those two types, we distinguish between full and partial migration:

- In the case of full migration an entire service stack is migrated, i.e., all components belonging to a certain service are migrated, e.g., a web server along with its database.
- Partial migration is linked to service disaggregation and describes the migration of service components or modules. A service composed of multiple components can be disaggregated into sub-services, some of which may then be migrated, before being reestablished as separate service.

IV. Towards a Reference Architecture

Cloud services offer access to services, which are associated with pools of stateful *resources*, e.g., virtual machines, data storage, queues, e-mail systems, etc. Our concept of a resource is similar to the notion of resources within the WS-Resource framework [20], however, less formalized because cloud services do not necessarily standardize on Web Service specifications.

We distinguish between two types of programmatic access to these resources:

- Resource API
- Management API

Applications implement the Resource API to access and utilize resources, which are exposed as business logic. For example, Amazon S3 offers a Resource API to create, read, update, and delete basic storage volumes ("buckets") as well as to upload or download data objects. Within the business logic of a photo-sharing application, buckets could be used as photo albums and a data object within a bucket could represent a photo image file. Table I illustrates that a photo-sharing application could be implemented with Cloud services from either Amazon or Google - or with a mix of services from both providers.

TABLE I
Example photo-sharing application.

Application feature	AWS	Google App Engine
Photo storage	S3 buckets & obj.	Data store or Blobstore
Photo notification	SQS or SNS	Channel service
Image editing	N/A	Image service
Photo sharing	SES	Mail service

The Management API helps application developers and administrators to manage resources efficiently. This includes a variety of activities: monitoring, deployment, data management, and so on. For example, Amazon EC2 offers a Management API for managing virtual machines (e.g., launch, stop, terminate) along with related settings and add-on services (e.g., security groups, block storage volumes, static IP addresses). Google App Engine offers a Management API to deploy application packages into the runtime environment and a dashboard for monitoring and administration (e.g., logs, cron jobs, datastore indexes, application versions and release management).

A. Two Perspectives on Interoperability

Interoperability challenges can be viewed from the perspective of a service provider or from the perspective of a service user. A service provider could be interested in offering distributed system services, which are interoperable with established, proprietary de-facto standards. Service users, on the other side, could design and implement applications with adaptors to multiple service providers, thereby enabling federation.

Table II shows 5 open source systems, which offer services similar to Amazon EC2 and Amazon S3. These systems support the interface definitions and protocols of their Amazon Web Services counterparts. Currently, all of the open source systems offer merely a small subset of comparable services and therefore only cover a subset of the Amazon Web Services API. The quality of a hosted open source solution, however, significantly depends on the system management skills of the hosting provider with regard to system scalability, performance and fault-tolerance. The systems could for example be backed with open source distributed system solutions, such as Apache HBase or Apache Cassandra, thereby providing a basis for achieving higher quality levels. Still, this induces even more integration challenges.

TABLE II
AMAZON WEB SERVICES (AWS) COMPATIBLE OPEN SOURCE CLOUD MANAGEMENT SYSTEMS.

Name	API	AWS-compatible services
Eucalyptus	AWS	Eucalyptus (EC2), Walrus (S3)
OpenNebula	OCCI, AWS	OpenNebula (EC2)
CloudStack	CloudStack, AWS	CloudStack + CloudBridge (EC2)
OpenStack	OpenStack, AWS	OpenStack: Compute, Image Service (EC2) & Object Storage (S3)
Nimbus	WSRF, AWS	Nimbus (EC2), Cumulus (S3)

TABLE III
COMPARISON OF MULTI-CLOUD LIBRARIES AND UTILITY PROGRAMS.

Name	Lang.	License	AWS[a]	RAX[b]	GOOG[c]	VMW[d]	MS[e]	GG[f]
jclouds	Java	Apache2	yes	yes	yes	yes	yes	yes
JetS3t	Java	Apache2	yes	no	yes	no	no	no
fog	Ruby	MIT	yes	yes	yes	no	no	yes
boto	Python	MIT	yes	no	yes	no	no	no
libcloud	Python	Apache2	yes	yes	no	yes	no	yes
deltacloud	Ruby	Apache2	yes	yes	no	no	no	yes
Whirr	Java	Apache2	yes	yes	no	no	no	no
PyStratus	Python	Apache2	yes	no	no	no	no	no

[a]Amazon [b]Rackspace [c]Google [d]VMWare [e]Microsoft [f]GoGrid

Table III shows a list of multi-cloud libraries, which enable interoperability across similar cloud services on a higher level than the systems discussed before. During the implementation of an application, the libraries jclouds, JetS3t, fog, boto, libcloud, and deltacloud are linked into the build path. When the application has been implemented, it can be deployed using any of the library-supported cloud services. This simplifies migration processes and redundancy setups as described in the sections before as there is no need to re-design the application. Instead, simple configuration options, usually just the service endpoints, must be changed. Figure 2 illustrates the migration of a service and the impacts on the service endpoints and the thereon based application.

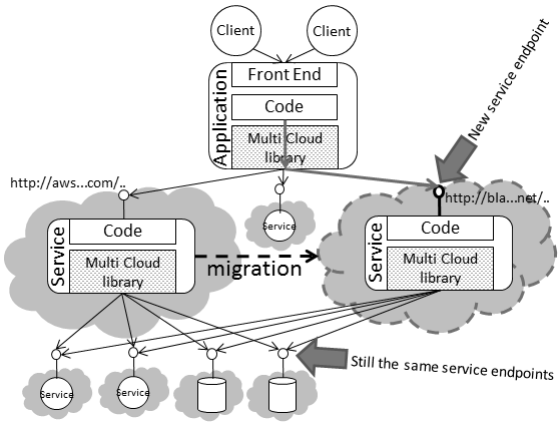

Fig. 2. Migration scenario illustrating impact on service endpoints

Additionally, utilities like Whirr (based on jclouds) and PyStratus can be used to deploy complex distributed systems on top of exchangeable compute clouds, such as EC2 or the Rackspace Cloud.

Both strategies, interoperable open source solutions and multi-cloud application code, can be employed to facilitate transparent application migration. Redundancy is more complicated to establish: Either it is explicitly foreseen in the application's code or there is a federation system providing a suitable programming abstraction. An additional layer decouples the application from the actual resources and permits their transparent reconfiguration, e.g., change redundancy strategy to erasure coding. Figure 3 illustrates the two strategies.

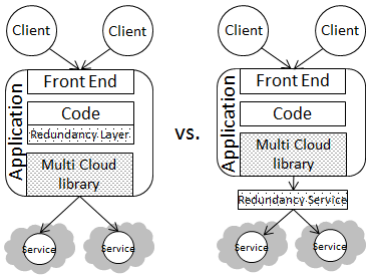

Fig. 3. Redundancy Strategies

B. Potential Reference Architecture Components

The open source cloud management systems in Table II and the multi-cloud software libraries in Table III are crucial elements for creating a federated cloud application. However, the systems and libraries should be discussed in a wider context to answer how applications can be migrated from one cloud service to another or operated on top of redundant cloud services.

We suggest that a reference architecture should contain the following components:

- **Provisioning Engine**: takes an application package along with policies and maps business logic components to a pool of resources. The projected mapping along with management configurations is then executed and enforced through a Distribution Manager.
- **Distribution Manager**: contains multiple subcomponents, through which it enforces guarantees specified with policies. For example, enforce consistency between data replica; enforce the same deployment configuration on multiple servers. It may also serve as a redundancy decoupling layer. Principal components are:
  - **Deployment Manager**: is a component of the Distribution Manager. Based on a deployment description the manager executes resource management commands through Resource Managers. It guarantees the avalabillity and the correct configuration of provisioned resources.
  - **Configuration Manager**: is a component of the Deployment Manager. It recreates virtual appliances

resp. application stacks based on stored configuration informations.

- **Data Distribution Manager**: is a component of the Distribution Manager. It manages the distribution of data, e.g., data replication, data redundancy, according to the distribution strategies.

The distribution managers secondary components are:

- **Transformation**: is used to transform incompatible formats, e.g., virtual machine images and to map between different data formats.
- **Monitoring**: gathers information about resource states and information about their configuration through the Resource Managers. In case of unexpected conditions the Distribution Manager adapts the system to match the projected provision mapping.

- **Resource Manager**: manages all resources in a unified way. It can be realized as a collection of resource-localized components. The Resource Manager provides an abstraction of the APIs of the underlaying services and allows the Distribution Manager to configure resources in different clouds in a unified way. It may use adaptors, for example multi-cloud libraries, to perform its tasks.

Figure 4 depicts our current vision of the reference architecture. We have to point out, that we are still doing research on the reference architecture and that the figure should only be considered a snapshot of our momentary work in progress.

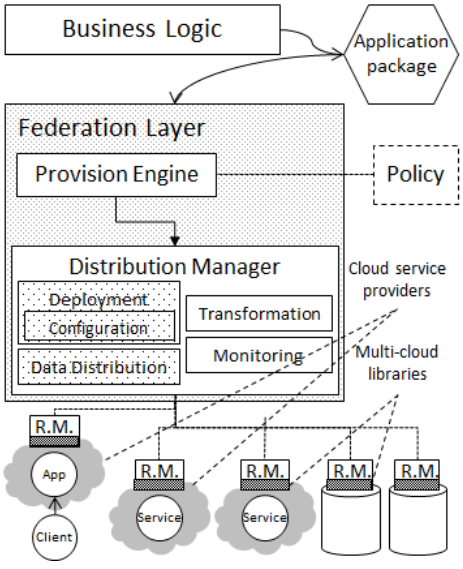

Fig. 4. Reference Architecture

V. Discussion

A. Vendor Lock-In and Cloud Computing

As depicted in Section II, vendor lock-in exists when potential switching costs surpass the benefits the customer would enjoy by switching to another provider. This is currently the case with Cloud Computing: by switching the provider the initial ex-ante investments could be largely lost and new investments, to adapt the software and retrain employees, will be necessary, thus exceeding the benefits of the provider-change. This implicates, that in Cloud Computing, lock-in, and in consequence hold-up, is a result of the different, proprietary interfaces, services and service offerings and the complexity involved in coping with this issues.

Since Cloud Federation resolves the above mentioned issues or - at the least - lowers the costs involved, we claim that it thereby resolves lock-in as well as hold-up and is a key enabler of Cloud marketplaces.

B. Future Work

Thoughts on Vertical and Secondary Services Federation are not incorporated in this article and will be subject to future works. Also the proposed federation reference architecture has to be elaborated in more detail in future works. Notably, we did not outline details on our vision of application packages and how the architecture's components could be realized.

VI. Conclusion

Cloud Federation is a concept, which has a large potential and might have an enormous influence on the way computing resources and applications will be handled, developed and used. It is a further step of providing computing resources in an utility-services-like way, similar to other services, e.g., electricity or water. However the evolution of Cloud Computing and related concepts and technologies is extremely dynamic and it is very difficult to make long-term prognoses. We believe anyhow, that this article can be a substantial contribution to future works on Cloud Federation.

Acknowledgment

The work presented in this paper was performed in the context of the Software-Cluster project EMERGENT [21]. It was partially funded by the German Federal Ministry of Education and Research (BMBF) under grant no. "01IC10S01". The authors assume responsibility for the content.

References

[1] R. Harms and M. Yamartino, "The economics of the cloud," Microsoft Corporation, Redmond, WA, USA, Microsoft Whitepaper, November 2010. [Online]. Available: http://www.microsoft.com/presspass/presskits/cloud/docs/The-Economics-of-the-Cloud.pdf

[2] "Microsoft: Smb hosted it commentary report," 2010.

[3] M. Armbrust *et al.*, "Above the clouds: A berkeley view of cloud computing," University of California at Berkeley, Tech. Rep., February 2009. [Online]. Available: http://berkeleyclouds.blogspot.com/2009/02/above-clouds-released.html

[4] P. Mell and T. Grance, "The nist definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, p. 50, 2009. [Online]. Available: http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

[5] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? an architectural map of the

cloud landscape," *Software Engineering Challenges of Cloud Computing, ICSE Workshop on*, vol. 0, pp. 23–31, 2009.

[6] C. Baun, M. Kunze, J. Nimis, and S. Tai, *Cloud Computing: Web-basierte dynamische IT-Services*, ser. Informatik im Fokus. Berlin: Springer-Verlag, Oktober 2009.

[7] R. Cowan, "Tortoises and hares: Choice among technologies of unknown merit," *The Economic Journal*, Jan. 1991. [Online]. Available: http://links.jstor.org/sici?sici=0013-0133(199107)101%253A407%253C801%253ATAHCAT%253E2.0.CO%253B2-S

[8] B. Klein, R. G. Crawford, and A. A. Alchian, "Vertical integration, appropriable rents, and the competitive contracting process," *Journal of Law & Economics*, vol. 21, no. 2, pp. 297–326, October 1978. [Online]. Available: http://ideas.repec.org/a/ucp/jlawec/v21y1978i2p297-326.html

[9] B. Holmstrom and J. Roberts, "The boundaries of the firm revisited," *Journal of Economic Perspectives*, vol. 12, no. 4, pp. 73–94, Fall 1998. [Online]. Available: http://ideas.repec.org/a/aea/jecper/v12y1998i4p73-94.html

[10] P. A. Grout, "Investment and wages in the absence of binding contracts: A nash bargining approach," *Econometrica*, vol. 52, no. 2, pp. 449–60, March 1984. [Online]. Available: http://ideas.repec.org/a/ecm/emetrp/v52y1984i2p449-60.html

[11] J. Tirole, "Procurement and renegotiation," *Journal of Political Economy*, vol. 94, no. 2, pp. 235–59, April 1986. [Online]. Available: http://ideas.repec.org/a/ucp/jpolec/v94y1986i2p235-59.html

[12] C. Ewerhart and P. W. Schmitz, "Der lock in effekt und das hold up problem," University Library of Munich, Germany, MPRA Paper 6944, 1997. [Online]. Available: http://ideas.repec.org/p/pra/mprapa/6944.html

[13] O. E. Williamson, *Markets and hierarchies: An analysis and antitrust Implications*. The Free Press, 1975.

[14] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10. [Online]. Available: http://portal.acm.org/citation.cfm?id=1251254.1251264

[15] J. Broberg, R. Buyya, and Z. Tari, "Metacdn: Harnessing 'storage clouds' for high performance content delivery," *J. Network and Computer Applications*, vol. 32, no. 5, pp. 1012–1022, 2009.

[16] K. D. Bowers, A. Juels, and A. Oprea, "Hail: a high-availability and integrity layer for cloud storage," in *ACM Conference on Computer and Communications Security*, 2009, pp. 187–198.

[17] D. Bermbach, M. Klems, M. Menzel, and S. Tai, "Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs," in *Proceedings of the IEEE Cloud 2011 - to appear*, 2011.

[18] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK: Springer-Verlag, 2002, pp. 328–338. [Online]. Available: http://portal.acm.org/citation.cfm?id=646334.687814

[19] Y. Saito, S. Frølund, A. C. Veitch, A. Merchant, and S. Spence, "Fab: building distributed enterprise disk arrays from commodity components," in *ASPLOS*, 2004, pp. 48–58.

[20] K. Czajkowski *et al.*, "The ws-resource framework," Tech. Rep. 1.0, March 2004. [Online]. Available: http://www.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf

[21] "Software-cluster emergent." [Online]. Available: www.software-cluster.org

APPENDIX

Table A - Cloud product overview

Name	URI
Amazon's EC2	http://aws.amazon.com/ec2/
Amazon's Elastic Beanstalk	http://aws.amazon.com/elasticbeanstalk/
Amazon's S3	http://aws.amazon.com/s3/
AppScale	http://code.google.com/p/appscale
CloudKick	http://www.cloudkick.com/
Google App Engine	http://code.google.com/appengine/
Microsoft Azure	http://www.microsoft.com/windowsazure/
SalesForce' Force.com	http://www.salesforce.com/platform/
ScaleUp	http://www.scaleupcloud.com/
SpotCloud	http://spotcloud.com
Zimory	http://www.zimory.com

Table B - Multi-cloud library overview

Name	URI
boto	http://code.google.com/p/boto/
deltacloud	http://incubator.apache.org/deltacloud/
fog	http://github.com/geemus/fog
jclouds	http://code.google.com/p/jclouds/
JetS3t	http://jets3t.s3.amazonaws.com/
libcloud	http://incubator.apache.org/libcloud/
PyStratus	https://github.com/digitalreasoning/PyStratus/
Whirr	http://incubator.apache.org/whirr/

Table C - Open source cloud management systems overview

Name	URI
CloudStack	http://cloud.com/
Eucalyptus	http://open.eucalyptus.com/
Nimbus	http://www.nimbusproject.org/
OpenNebula	http://opennebula.org/
OpenStack	http://www.openstack.org/