Cloud Federation: Effects of Federated Compute Resources on Quality of Service and Cost

David Bermbach
Karlsruhe Institute of Technology
Karlsruhe, Germany
david.bermbach@kit.edu

Tobias Kurze
Karlsruhe Institute of Technology
Karlsruhe, Germany
tobias.kurze@kit.edu

Stefan Tai
Karlsruhe Institute of Technology
Karlsruhe, Germany
stefan.tai@kit.edu

***Abstract*—Cloud Federation is one concept to confront challenges that still persist in Cloud Computing, such as vendor lock-in or compliance requirements. The lack of a standardized meaning for the term Cloud Federation has led to multiple conflicting definitions and an unclear prospect of its possible benefits.**

Taking a client-side perspective on federated compute services, we analyse how choosing a certain federation strategy affects Quality of Service and cost of the resulting service or application. Based on a use case, we experimentally prove our analysis to be correct and describe the different trade-offs that exist within each of the strategies.

I. Introduction

Over the last few years Cloud Computing has become more and more adopted and most companies at least consider some kind of cloud strategy. Typical motivations are potential cost reductions, variabilization of fix costs, increased availability through geographic distribution and, hence, improved resilience to geographically limited faults, flexibility and scalability.

However, these benefits come at a price: A customer is typically tied closely to just one provider, when leveraging the full potential of all provided features. This, of course, implies a high degree of vendor lock-in where the customer not only depends on the proprietary interfaces and APIs specified by the provider but also employs a workforce of specialists for that particular provider who are unlikely to be able to work with a competitor's cloud offering.

Another issue is trust: An IT infrastructure running in the cloud is beyond physical control of the customer as servers are owned and operated by the provider. Security and compliance with data privacy laws are also of concern. Surveys (e.g., [1]) show that this poses a problem for quite a few potential customers who still hesitate to use cloud computing.

This all, combined with the fact that using a single provider introduces a single point of failure into one's IT landscape [3], can lead to underinvestment – a situation described as the hold-up problem [9]. As such a situation is not pareto-optimal it is desirable to address these concerns.

Cloud Federation [2], [4], [5], [12] has been proposed as a way to address many of these problems. So far, there is an abundance of work where federated storage services are used and different cloud federation strategies have been identified but it is relatively unclear which quality of service (QoS) trade-offs exist between those strategies and how choosing such a strategy may affect the amount of money spent on cloud resources. Though, most people will be able to name qualitative trade-offs but quantifying these QoS effects is an entirely different story. With this work we hope to shed some light onto the qualitative trade-offs and try to verify whether common assumptions are actually true. In this work, we do this for federated compute resources.

This work is structured as follows: In section II we describe different compute federation strategies and analyze how they affect select QoS attributes. Afterwards, in section III, we verify our findings based on a use case which we apply to all strategies before discussing trade-offs that we have identified in section IV and comparing it to related work in section V. Finally, we conclude this work with section VI where we revisit our observations and point out future work.

II. Background

In [12] we defined cloud federation as using more than one cloud service at the same time. Also, we proposed to distinguish cloud federation along several dimensions:

1) Federation can occur within a layer of the technical cloud computing stack [13] (i.e., as an example to federate two infrastructure services) or across layers (e.g., a platform as a service (PaaS) offering combined with an Infrastructure as a Service (IaaS) offering).
2) For federation within the IaaS layer there are two main kinds of services: compute and storage.
3) Cloud federation can happen for an unlimited (at least regarding a near time horizon) or a limited period of time. The first case is typically called redundancy while the latter is called migration.

In this work, we want to focus on the redundant usage of compute resources, i.e., on permanent federation of services within the IaaS layer. We believe, though, that our results can

easily be transferred to migration scenarios. In [12], we have identified five strategies for the redundant usage of compute services which we will analyse in this section.

Furthermore, there are two main classes of application workloads which differ in the way requests are created. While in batch processing (also known as OLAP), as the name already suggests, requests typically arrive in batches, in customer facing applications (also known as OLTP) requests are triggered one by one by a client. In the following, we will focus on batch processing scenarios but we believe that OLTP applications can be described similarly although there might be additional requirements that need to be considered.

A. Redundant Deployment (RD)

In this strategy, the same application logic is deployed with different providers and usually also geographically distributed. Then, requests are routed to one of the machines which processes the request and responds afterwards to the client.

The main motivation behind this approach is to remove the single point of failure which exists in a scenario that uses only one provider. This, of course, positively affects availability. Furthermore, this approach is likely to improve response times due to increased client proximity of the deployments. Another motivation can be compliance with laws and regulations which sometimes require to keep sensitive data within a country.

RD offers a very high level of availability and an excellent resilience to failures not caused by the application code itself. It can be calculated as

$$A = 1 - \prod_i (1 - A_i)$$

where A is the resulting availability of all deployments combined and A_i is the availability of deployment i. Concerning scenarios that make requirements on request handling, such as compliance with laws and regulations, availability only refers to a subset of clouds, matching the demanded requirements.

Another aspect is the part of the batch p_i that is processed by a particular cloud i which is identical to the probability of a particular request being processed in that cloud. If t_i denotes the mean value of the processing time per request in cloud i, then we can calculate the probability of processing a particular request in cloud i as

$$p_i = \frac{\prod_{j \neq i} t_j}{\sum_{m=1}^{n} \prod_{k \neq m} t_k}$$

where n is the number of clouds involved. Using this probability, we can deduce the expected value of the average processing time as

$$T = \sum_i p_i * t_i$$

Regarding cost, this strategy should be cost neutral compared to having several instances with the same provider as every instance is used as efficiently as possible. For long term contractual commitments, though, this strategy can boost cost if only small batches need to be processed.

B. Redundant Computation (RC)

In RC the same application logic is deployed to different providers. In contrast to RD, though, every request is forwarded to all clouds. Depending on the goal of this strategy, we can then either wait for all results and compare them (RC with Comparison) or use the first result which is returned (RC without Comparison).

The latter helps to improve request latencies and reduces the chance of a request timing out due to a crashed machine, an approach also taken in Google's MapReduce [8]. The first strategy, in contrast, is useful if we do not particularly trust a certain provider or if we fear that a machine might be corrupted by a third party.

Regarding cost both RC strategies add a huge overhead as all data is processed on all instances.

1) Redundant Computation with Comparison (RCC): This scenario provides rather poor availability – the more participating clouds, the lower the availability. It can be calculated as

$$A = \prod_i A_i$$

and all requests are processed in all clouds, i.e., the probability of processing a particular request in cloud i is 100%. The distribution of resulting processing times is basically a convolution of all participating clouds' processing time distributions using the max operator.

2) Redundant Computation without Comparison (RCwC): In contrast to RCC, this scenario provides excellent availability and performance – the more participating clouds, the better availability and response times. While the availability is identical to the RD scenario, the distribution of resulting processing times is again a convolution of all participating clouds' processing time distributions, this time using the min operator. This leads to high monetary costs as all requests are processed in all clouds.

C. Parallel Computation (PC)

In PC the same or very similar application logic is deployed to different providers. Incoming requests are broken down at bit level so that each cloud processes only a subset of the data. If subsets overlap, PC also allows validation of correctness of the answers like in RCC.

There are two main goals for this scenario: First, for sensitive data, processing only fragments guarantees that a malevolent cloud provider sees only parts of the data. Second, a fragment is smaller than the original data so that processing becomes faster and PC can, hence, be expected to complete faster than all other strategies. Processing only

fragments is also the main idea behind MapReduce [8]. Of course, not every request can be processed on fragments of data.

The distribution of the resulting processing time is again a convolution of the participating clouds' processing times using the max operator and is, hence, vulnerable to hung requests.

PC should generally be cost neutral as long as fragments do not overlap.

1) Secure Parallel Computation (PCS): In this scenario data is not only broken down at bit level but is also encrypted. While this adds a small overhead, most of the benefits of PC regarding performance persist. A downside, though, is that there are only very few operations that can be executed on encrypted data. Furthermore, the resulting availability is again the product of all participating availabilities.

2) Insecure Parallel Computation (PCI): This scenario is very similar to PCS regarding performance and availability. The difference is, that there are many more operations that can process fragments of data compared to operations that can process encrypted fragments. For example, it is still possible to count faces in images when allowing overlapping fragments. The first cloud could process the lower two quarters, the second one the upper two quarters and yet another one could process the middle two quarters.

III. Evaluation

To evaluate whether our analytical results can also be seen in experiments, we searched for a simple use case that works for *all* scenarios. We decided on a service which takes a random image as input and returns its monochrome version. Since this affects only a single pixel at a time, this use case is compatible with all strategies, even with PCS where pixels are scrambled randomly using a one time pad before fragmenting the scrambled image. Similar real-world use cases that are compatible with most strategies (i.e., all but PCS) could be face recognition in videos, pattern matching for large text files, video encoding etc.

A. Experimental Setup

For testing we used an Amazon EC2 m1.xlarge instance[1] in the region eu-west (Ireland) and an 1&1 Dynamic Cloud Server [2] with 2 cores and 3GB RAM. We then measured the distribution of processing times using just a single cloud deployment before benchmarking all compute federation strategies in a two cloud deployment. For each test run we issued at least 50 requests from a standard laptop computer, to simulate an on premises server, and sent it to one or both cloud servers (depending on the strategy). Every request contained the same image. We repeated all test runs several times without significant changes in the results[3].

Client-side load balancing strategies depended on the particular strategy that was benchmarked: For RD, all requests were sent to a (local) queue and the cloud servers dequeued the images one after the other. So, effectively both servers were working in parallel at maximum resource usage without any idle times. For both RC strategies a request was broadcast to both servers at the same time. The subsequent request was then sent after the first (or the second respectively for RCC) server completed its request. For both PC strategies, the images were fragmented and put into two dedicated queues – one per server. Each cloud server then pulled its requests from the corresponding queue, so that both servers were effectively running in parallel. Note, that both servers had to terminate at the same time, i.e., the faster server incurred an idle time (in the end) of the difference between both clouds' total processing times.

After all test runs had been completed we used the single cloud data as input for the formulas from section II and compared it to the the results from the two cloud benchmarks. So, in the following,

- *experimental data* describes the benchmarking results from our two-cloud deployments and
- *simulation data* is created using Monte-Carlo simulations of the two single-cloud deployments' benchmarking results and the formulas from section II.

As during the original test run most of the processing time was spent on network IO, we also ran two variants of the test where we added artificial delays on the server side: the second scenario tried to compensate for the fact that both the 1&1 server as well as the laptop computer were both situated in Karlsruhe, Germany and added 500ms delay to the Amazon instance and 1800ms to the 1&1 instance. During initial test runs, this "delay configuration" resulted in fairly similar total processing times for both clouds. The third scenario, finally, even exceeded the second one by having no delay for Amazon and fully two seconds for 1&1. For the remainder of this paper, we will refer to those setups as scenarios 1, 2 and 3. Our target was for scenario 1 to have 1&1 run faster, for scenario 2 to have both clouds comparably fast and for scenario 3 to have AWS run faster.

Please, note, that for our purposes neither the actual performance of the clouds nor the performance differences between both clouds matters. Our sole aim was to simulate different compute instances and to evaluate whether our observations from the previous section are correct.

[1] aws.amazon.com/ec2

[2] hosting.1und1.de/CloudDynamicServer

[3] The actual performance varied over time up to 25% but did so consistently for single cloud deployments as well as for the federation tests so that in the end the ratio of benchmark and simulation results was unaffected.

B. Findings

In the following, we show how experimental results for all federation strategies compare to simulated results based on single cloud processing times distributions and the equations from section II.

1) Availability: None of our test runs encountered any kind of inavailability of the cloud services. Hence, we cannot use real world observations for proving the correctness of our considerations. Instead, we artificially introduced failures (i.e., killing the process on one of the servers) and in every case the system showed the expected behavior, i.e., all strategies apart from RD and RCwC became unavailable and those two showed slightly poorer performance depending on which instance was terminated.

2) Processing Time: While some simulation results for the average processing time are highly accurate (especially for RD and RCC) others show some larger deviations, see Table II for details. One of the reasons is the time difference between measuring single cloud values versus the corresponding federation value for the different strategies. As already mentioned above we saw random fluctuations in processing times up to 25% for both single cloud and federated deployments over a period of a few minutes. The tests showing the highest degree of matching between simulation and actual data are the ones where the single cloud tests were run right before or after the federated tests. Hence, our simulations reached an accuracy level with less then 5% error.

For the PC strategies' simulation we used half the values measured during the initial single cloud deployments which does neither consider the overhead for fragmenting the data nor the overhead for establishing a connection. As a result, our simulations were off by about 18% respectively 19% for scenario 1 but got closer with about 2% error for scenario 2 and less than 1% for scenario 3. This is due to the fact, that the artificial delay introduced stays constant no matter how much deviation the actual data transfer and computation durations show. Hence, the relative error decreases. Due to fluctuations of e.g., network load during daytime, PCS performed slightly better than PCI, even though PCS inflicts an additional overhead. See table II for more details..

The same goes for RCwC which shows the same development of relative deviations. Scenario 1 starts off with a deviation of almost 11%, though, which can be explained by the high performance of this strategy where small absolute deviations amount to large relative errors. Nevertheless, we believe that our results prove our considerations from section II to be correct.

Comparing the different strategies, as expected RD runs fastest as both servers reduce the number of jobs in parallel while RCC is the slowest strategy. RCwC is somewhere in the middle.

PC does not fit in as easily: For scenario 1 this is relatively straightforward as both servers process only half the data but the slowest one determines the progress speed. As expected, the total of two times the processing time for PC minus a small overhead equals approximately RCC. For the other two scenarios comparison is more difficult: Table II shows the processing times for running the jobs with the full delays each, even though each server processes only half the data. The processing times include network latency, and basically describes the time from when a request was sent till an answer is received. So, it might be fairer to use only half the delays (e.g., 250ms and 900ms for scenario 2) but this depends on what the delays stand for. Full delays make sense when the delays are for example the routing overhead of some federating component (i.e., incurred on premises and not in the cloud or on the way to there). Another example could be a manual job approval process for each task that applies fully to each request. Half the delays make sense when they stand for slow processing or data transfer according to the linear complexity of our sample usecase. Still, there are usecases with non-linear complexity where even other fractions of the delays might make sense.

Table I
JOB DISTRIBUTION USING REDUNDANT DEPLOYMENT

Scenario	AWS	1&1
Scenario 1: Simulation	16.02%	83.98%
Scenario 1: Experiment	16.00%	84.00%
Scenario 2: Simulation	54.88%	45.12%
Scenario 2: Experiment	54.00%	46.00%
Scenario 3: Simulation	63.04%	36.96%
Scenario 3: Experiment	64.00%	36.00%

3) Probability of Processing in Cloud i: In a RCC or PC setup every request is processed by every participating instance. Therefore, the probability that a request is processed in Cloud i equals 1 for all i.

The same applies to RCwC where all requests are at least sent to all instances, though, the requests may not be fully executed. We were able to observe exactly the expected behavior during all our test runs.

For RD the matter is more complicated. As stated before, we used a single queue from which jobs were submitted to two instances as our test setup. Since the two instances had different configurations in terms of virtual hardware as well as network connectivity, we observed an unequal distribution of jobs between the clouds. Depending on the scenario, either the AWS instance or the 1&1 instance handled a larger share of the jobs. Using a Monte-Carlo simulation we got a very good approximation of the actual experimental distribution of jobs. Table I gives an overview of the test and the simulation results.

4) Cost: To calculate the costs of our test setup, we considered every aspect that may affect costs, such as instance hours and network traffic charges or free tiers. A huge difference between the cost models of Amazon and

Table II
PROCESSING TIMES FOR DIFFERENT FEDERATION STRATEGIES COMPARING RESULTS FROM SIMULATION AND EXPERIMENT

	Scenario 1			Scenario 2			Scenario 3		
Strategy	Exp.	Sim.	Error	Exp.	Sim.	Error	Exp.	Sim.	Error
RD	283ms	283ms	0.23%	970ms	942ms	-4.69%	846ms	855ms	1.01%
RCC	1801ms	1722ms	-4.33%	2088ms	2054ms	-1.59%	2286ms	2300ms	0.63%
RCwC	384ms	342ms	-10.81%	1772ms	1683ms	-5.03%	1622ms	1636ms	0.84%
PCS	1057ms	863ms	-18.34%	1967ms	1927ms	-2.02%	2170ms	2175ms	0.26%
PCI	1067ms	863ms	-19.04%	1968ms	1927ms	-2.09%	2195ms	2175ms	-0.88%

1&1 is that 1&1's Dynamic Cloud server has a monthly subscription model. There is no pay-as-you-go alternative yet, though, in some long-running scenarios it might be less expensive to use fix-cost based offerings. Amazon also offers so-called reserved instances which come at fixed reservation price plus a reduced variable rate for the actual usage hours. We considered this in our cost calculations and chose the cheapest alternative respectively.[4]

As expected, RD is the most cost-efficient strategy. It is also the only strategy that, for 10 million images, is still less expensive when using Amazon on-demand instances instead of reserved instances. RC strategies profit more or less considerably from using reserved instances – at least for such a huge batch size. Figure 1 show a cost comparison of the different strategies for all three scenarios.

On the other end, RCC is the most expensive strategy as all requests are executed on all instances. Hence, the slowest instance determines how long all servers need to be running. RCwC on the other hand depends on the performance differences between the servers. For example, in scenario 2, where both servers show almost identical performance, this strategy costs almost the same as RCC. Basically, the cost can be approximated as the cost for having the fastest server process all requests plus the cost of having all other servers running idle (when ignoring cost for traffic etc.).

For PC, the calculation is a bit more difficult in terms of fairness regarding the delays as we already pointed out in our discussion of the processing times. One thing can be seen independent of the choice of delays: Security adds another small overhead in terms of cost which is not surprising considering the fact that it directly adds a processing time overhead.

IV. DISCUSSION

In section III we proved the formulas from section II to be correct and provided simulated as well as experimental results. We also indicated the theoretical costs involved with different federation strategies and scenarios. For our analysed QoS properties it really depends on the chosen federation strategy whether there exists a true monetary trade-off or whether spending more money actually proves counter-productive as there are other more hidden trade-offs involved. In the following we will analyse these trade-offs for each of the strategies.

A. Redundant Deployment

In RD adding more servers increases the total availability and improves the expected processing time. So, there is a direct trade-off between cost and availability as well as cost and processing time. Once, certain availability and processing time requirements have been identified it is easily possible to choose an optimal set of providers, e.g., using the methods of Menzel et al. [15]. This strategy does not increase correctness of results or security. Rather it might corrupt the latter as using multiple providers increases the risk of a malevolent provider.

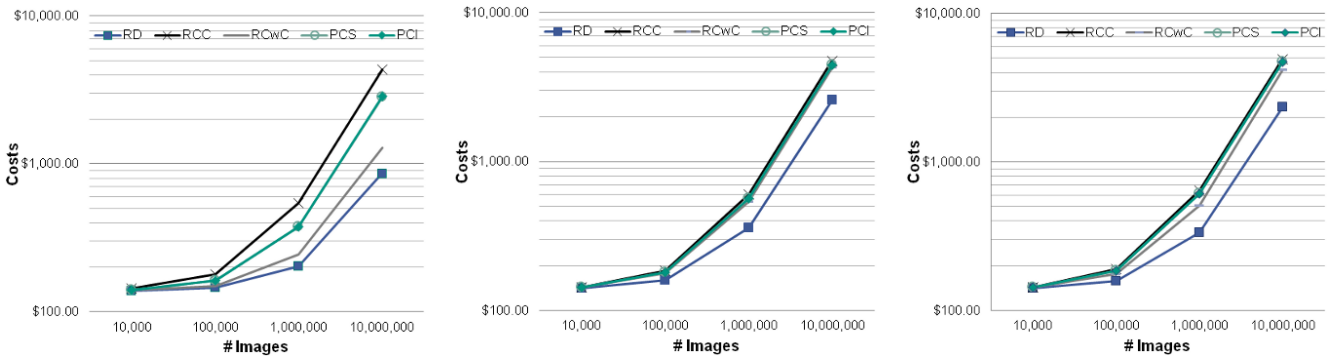
B. Redundant Computation with Comparison

Here, using more providers and, hence, adding more servers actually decreases the availability of the service and increases the processing times. Also, adding more providers again increases the likelihood of a malevolent provider, thus, potentially corrupting security. At the same time, this is the only federation strategy which allows to increase correctness (or rather the probability of having correct results). So, depending on the required level of correctness the minimum number of providers fulfilling that requirement should be chosen.

C. Redundant Computation without Comparison

Following this strategy, using more providers both increases availability as well as speeds up processing times. It is especially useful against peaks for single requests due to a hung instance or similar reasons as there is always at least one backup. The price one has to pay for this, though, is high in terms of monetary cost as having multiple instances complete the same task is completely inefficient. So, there is a direct trade-off between availability, resilience to processing time peaks as well as processing time (though weaker than with RD) and cost. Like in RD, this strategy does not affect correctness of results but might negatively affect security.

[4]For cross-currency conversion we used the EUR-USD exchange rate of February 9, 2012 and observations are based on the available offerings and prices of the same date.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 1. Calculated costs for scenarios 1 to 3

Table III
INFLUENCE OF FEDERATION STRATEGIES ON QOS

	QoS Characteristics				
Strategy	Availability	Security	Performance	Correctness	Cost
RD	++	-	++	o	++
RCC	--	--	--	++	--
RCwC	++	--	+	o	-
PCS	-[1]/+[2]	++	-	o[1]/+[2]	++
PCI	-[1]/+[2]	+	o[1]/-[2]	o[1]/+[2]	++

[1]Disjunct requests
[2]Overlapping requests

D. Parallel Computation (Insecure)

Here, adding more providers improves processing times. Depending on whether fragments overlap, it also increases (if they do) or decreases availability (if they do not). Again, if parts overlap this can also improve correctness of the results and security is positively affected. So, if fragments overlap, it is only a matter of money to reach the desired QoS levels. If not, there is a trade-off between the desired level of security and processing performance against availability and cost. Furthermore, this strategy is not applicable to all use cases.

As a side note: PCI is partly a misnomer as it actually increases security compared to a single cloud deployment. We nevertheless chose the name to point out the difference to PCS where additional encryption is used.

E. Parallel Computation (Secure)

For this strategy, the same holds as for PCI. The only changes are that there is a very small monetary surcharge for creating near perfect security as well as a small processing overhead. Also, there are even less use cases where this strategy can be used.

Table III gives an overview how each of the federation strategies affects the discussed QoS characteristics compared to a non-federated deployment.

V. RELATED WORK

In [12] we focused on a customer perspective and analyzed federation strategies accordingly, i.e., pointed out their advantages from a customer's point of view to avoid vendor lock-in or increase security, for example. This fact was also reflected in our vision of a potential cloud federation architecture.

In contrast to our approach, there are some alternatives that focus on the provider's perspective. [10], [11] propose federated clouds to enable the in- and outsourcing of workloads between different clouds in order to enhance provider profit. To establish such as system, a global scheduling layer is proposed and certain components have to be in place at the providers' sites. So-called federated cloud providers are able to interact and to exchange workload.

Another federation approach that aims to add new functionalities to cloud providers has been proposed by Celesti et al. [7]. In the described cross-cloud federation scenario a cloud operator is able to request further computing and storage capabilities from other clouds, allowing to satisfy additional service requests via a so-called Cross-Cloud Federation Manager placed inside cloud architectures.

A broker-based federation approach has been proposed by Villegas et al. [17]. Incoming customer requests sent to one cloud might be fulfilled by another cloud, mediated by a brokering structure. One aspect of the proposed model is that federation occurs between cloud providers at matching layers of the service stack. This enables isolation between brokering strategies at different layers of the cloud stack.

Zimory [5] offers a cloud marketplace where excess datacenter capacities can be offered as cloud computing resources. This is similar to the broker-based approach of Villegas et al. [17] but aims to hide the actual federation.

The PaaS offering Cloud Foundry [6] supports deployment on several infrastructure clouds [7]. So far, to our knowledge,

[5]zimory.com
[6]cloudfoundry.com
[7]gigaom.com/cloud/cloud-foundry-lets-apps-span-cloud-providers

only the Redundant Deployment strategy has been implemented.

This paper is focused on the client-side federation of compute resources where to our knowledge exist only very little work. Other areas like federated storage have been covered extensively, though, e.g.: [2], [4]–[6], [14], [16].

VI. Conclusion

This paper is intended as a logical successor to our previous work [12]. Hence, we started with a short recap of the five compute redundancy strategies before analysing how these strategies each affect availability, processing time, job distribution on different clouds, correctness of results, security and cost. Where applicable, we presented formulas to calculate or approximate the degree of quality.

Next, we measured each quality level for single cloud deployment as well as for each federation strategy using a usecase in three different scenarios. Based on these results we then showed that our formulas from section II are valid.

As a final contribution, we used our experimental observations to point out the main trade-offs in the five federation strategies. To our knowledge there is no prior work that takes a client-side perspective on federation of compute services in the cloud which we pointed out when discussing related work.

Future work should develop best-practices, reference architectures and corresponding frameworks for actually implementing the different federation strategies further extending previous considerations in [12]. Other relevant questions could be how to decide on a particular strategy or how to change a running system from a single cloud deployment or a federated deployment to another federation strategy.

As we pointed out, different kinds of failures can occur, ranging from software failures to network partitionings. Future work should analyse how different failure scenarios affect the QoS levels for each of the federation strategies. Finally, all future work we just discussed should be transferred to storage services (though there already is some work on federated storage) and it is still unclear how cloud federation is possible for both the PaaS and the SaaS layer.

Acknowledgment

We would like to thank Amazon Web Services and 1&1 who provided cloud resources for our experiments.

References

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

[2] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. RACS: a case for cloud storage diversity. In *SOCC*. ACM, 2010.

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, H. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. Above the Clouds: A Berkeley View of Cloud Computing. UC Berkeley, 2009.

[4] D. Bermbach, M. Klems, M. Menzel, and S. Tai. Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs. In *IEEE CLOUD*. IEEE, 2011.

[5] K. Bowers, A. Juels, and A. Oprea. HAIL: A high-availability and integrity layer for cloud storage. In *CCS*. ACM, 2009.

[6] Broberg, Buyya, and Tari. Creating a Cloud Storage Mashup for High Performance, Low Cost Content Delivery. In *ICSOC 2008 Workshops*, pages 178–183. Springer, 2009.

[7] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. How to enhance cloud architectures to enable cross-federation. *IEEE CLOUD*, pages 337–345, 2010.

[8] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[9] C. Ewerhart and P. W. Schmitz. Der lock in effekt und das hold up problem. MPRA Paper 6944, University Library of Munich, Germany, 1997.

[10] I. Goiri, J. Guitart, and J. Torres. Characterizing cloud federation for enhancing providers' profit. In *IEEE CLOUD*. IEEE, 2010.

[11] Í. Goiri, J. Guitart, and J. Torres. Economic model of a cloud provider operating in a federated cloud. *Information Systems Frontiers*, pages 1–17, 2011.

[12] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze. Cloud federation. In *CLOUD COMPUTING 2011*, pages 32–38, 2011.

[13] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm. What's inside the Cloud? An architectural map of the Cloud landscape. In *ICSE 2009 Workshops*, pages 23–31. IEEE, 2009.

[14] I. Livenson and E. Laure. Towards transparent integration of heterogeneous cloud storage platforms. In *DIDC*. ACM, 2011.

[15] M. Menzel, M. Schönherr, J. Nimis, and S. Tai. $(mc^2)^2$: A generic decision-making framework and its application to cloud computing. In *CCV*, Singapore, Mai 2010. GSTF.

[16] S. Sakr, L. Zhao, H. Wada, and A. Liu. Clouddb autoadmin: Towards a truly elastic cloud-based data store. In *ICWS*, pages 732–733. IEEE, 2011.

[17] D. Villegas, N. Bobroff, I. Rodero, J. Delgado, Y. Liu, A. Devarakonda, L. Fong, S. M. Sadjadi, and M. Parashar. Cloud federation in a layered service model. *Journal of Computer and System Sciences*, 2012.