RZ 3783 (# Z1010-001) 10/21/2010
Computer Science 6 pages

Research Report

Dependable Storage in the Intercloud

C. Cachin, R. Haas, M. Vukolić

IBM Research – Zurich
8803 Rüschlikon
Switzerland

Dependable Storage in the Intercloud*

Christian Cachin
IBM Research - Zurich
cca@zurich.ibm.com

Robert Haas
IBM Research - Zurich
rha@zurich.ibm.com

Marko Vukolić
IBM Research - Zurich
mvu@zurich.ibm.com

Abstract

The cloud computing paradigm with its elastic pay-as-you-go model, "infinite" scalability and "always-on" availability arguably changes the landscape of services and systems. However, so far, cloud proliferation has not lived up to expectations in the enterprise segment. Often cited issues include confidentiality and integrity, but also reliability and consistency.

In this paper, we argue for the *Intercloud* as the second layer in the cloud computing stack, with the goal of building more dependable cloud services and systems. In the Intercloud layer, we foresee client-centric distributed protocols to complement more provider-centric, large-scale ones in the (Intra)cloud layer. These client-centric protocols orchestrate multiple clouds to boost dependability by leveraging inherent cloud heterogeneity and failure independence.

We also discuss the design of *Intercloud storage*, which we currently are implementing, as a primer for dependable services in the Intercloud. Intercloud Storage precisely addresses and improves the CIRC attributes (confidentiality, integrity, reliability and consistency) of today's cloud storage services.

1 Introduction

Cloud computing promises to the client an elastic pay-as-you-go model, "infinite" scalability and "always-on" availability, which renders it appealing for data and computation outsourcing, both for consumers that want to share their pictures with friends and for enterprises that want to reduce their IT budgets.

However, there are obvious dependability and security concerns associated with outsourcing data and computation to a potentially untrusted third-party (cloud). Even if the cloud provider is itself trusted by the client, issues like multi-tenancy entail vulnerabilities. More specifically, often cited problems include data confidentiality and integrity, but also reliability and consistency of the contracted service (we collectively refer to these four dimensions as CIRC).

We believe that a promising solution for improved cloud security and dependability lies in the *Intercloud*[1], the cloud of clouds, and goes beyond adding perfection to single, isolated clouds. In this paper, we first argue for the Intercloud as the second layer in the dependable cloud computing stack for the next-generation cloud. The upcoming Intercloud and today's single-provider clouds are two separate layers in the cloud computing stack of tomorrow (Fig. 1) that complement each other. The Intercloud offers promising solutions for enhanced dependability. Secondly, we present the design of a service in the Intercloud, which exploits the unique features of this model: An *Interclud storage (ICStore)* service that is currently under development and addresses the CIRC dimensions through a layered architecture (Sec. 3).

2 Dependable Cloud Computing Stack

In this section, we first overview the single-domain cloud layer and its dependability limitations and then argue for looking at Intercloud for solutions.

2.1 Single-Domain Cloud and Limitations

This layer encompasses most of the cloud computing systems to date, and consists of distributed protocols designed to run in a single administrative domain, typically under the control of *one* service provider (e.g., Amazon AWS[2], Google Apps [3], Nirvanix[4]). Distributed protocols used in this context are intended for wide-area systems,

*IBM Research Report RZ 3783, May 28, 2010.

[1]http://www.google.com/search?q=intercloud
[2]http://aws.amazon.com/
[3]http://www.google.com/a/
[4]http://www.nirvanix.com/

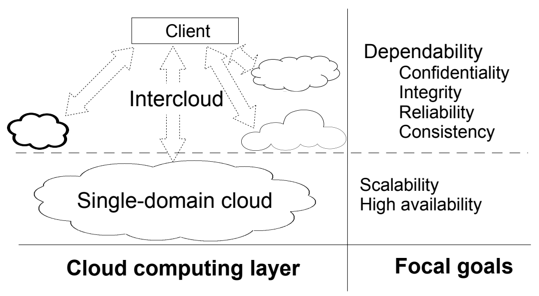

Figure 1: Dependable cloud computing stack.

with scalability to a very large number of clients and high availability as their most important goals [18].

The dependability and security of a single-domain cloud, especially its integrity, confidentiality, and isolation for data and computations in a multi-tenant model, are receiving increased attention (e.g., [5, 16]). However, devising a dependable service while relying on offerings of a single cloud provider P has its inherent limitations, since all trust in the system reduces to trusting P. Consider for example, encryption as the classical way for achieving data confidentiality. Encryption creates keys to be managed: but if a client solely relies on offerings of provider P, it would have to store encryption keys through a service offered by provider P as well, which immediately defeats the benefits of encryption. On the other hand, we consider storing the encryption keys locally at fault-prone clients to be an unacceptable solution, since losing a key implies losing the encrypted data. Moreover, when data and computation are outsourced to an untrusted cloud, integrity of data and computation can be guaranteed only to a certain extent [7].

Other limitations in relying on a single cloud provider's services are related to data reliability and consistency. While clouds are designed to be highly available, outages may and do occur at any individual provider (see, e.g., [1] for details). In addition, the network to a cloud provider remains a single point of failure, especially in the case of cloud providers that do not geographically diversify their services. Moreover, network connections are particularly vulnerable when the client resides outside North America and Western Europe, where high-bandwidth connections might not be readily available. Finally, eventual consistency offered by many cloud providers, which comes as a byproduct of high availability goals [18], might not be sufficient for some applications. Single-cloud solutions may give an incentive for a client to locally cache data, in order to avoid consistency problems. But this complicates concurrent access to the service or may completely defeat the very purpose of outsourcing data to the cloud.

To cope with these and other single-cloud dependability issues, we propose to look at the Intercloud.

2.2 Intercloud Layer

We advocate to leverage a multitude of cloud providers for addressing dependability issues and limitations of isolated single-domain clouds. These multiple cloud providers form the *Intercloud*, which we expect to become an increasingly relevant layer in the cloud computing model at large, which resides on top of the single-domain cloud layer.

The Intercloud layer offers a unique environment for building dependable services. Consider, for example, fault-tolerance as one of the key aspects of dependability. A fundamental assumption, on which virtually all fault-tolerant dependable systems rely, is the *failure independence* assumption. This assumption comes in different flavors, ranging from classical threshold failure assumptions to non-threshold failure models (e.g., [12]). In the first case, failure independence is reflected through the assumption that only the number of faulty processes counts, i.e., up to t but not $t + 1$ or more processes may fail; in the second case, failure independence is assumed across different fault-prone sets. However, the coverage of this assumption in practice is sometimes small. This remains a weak point of many dependable systems, and results in the assumption often being criticized. In this aspect, the Intercloud is without precedent: it comes with diversity in geographical locations, power supplies, internal cloud architectures, separate administrative domains and different proprietary middleware and application implementations. Best of all, this diversity comes to the client essentially for free, because the maintenance of such diversified (yet semantically similar) services is not the responsibility of a client, but is in the hands of the separate cloud providers. Since cloud-provided resources are a commodity, their cost is absorbed by many clients for different tasks. In contrast, maintaining enough diversity for a single task "in-house," which could include (but is not limited to) hardware and operating systems diversity, n-version programming and software maintenance, and administration know-how, is prohibitively expensive. Furthermore, trusting the ensemble of diversified clouds by distributing trust across different cloud domains is an appealing alternative to trusting (possibly critical) data to a single cloud provider.

Clearly, the Intercloud layer does not replace the single-cloud layer, but greatly expands its scope. We expect that dependable protocols in the Intercloud will be client-centric first, where client-side proxies orchestrate multiple clouds. This will be followed by more sophisticated services involving communication among different cloud services (this is not easily possible today due to lack of standardization). We envisage that protocols in the Intercloud add considerable value over the single cloud, because they maximize various QoS metrics, including dependability in the broad sense, and CIRC in particular. Of course, large-scale provider-centric single-domain clouds will continue to become more dependable; but Intercloud protocols are orthogonal to these efforts and directly benefit from them.

Intercloud applications already exist as mash-ups, which are used to build new web services leveraging applications from multiple providers (e.g., using Google Maps to geo-tag other web services, not necessarily related to Google). However, with respect to dependability, the Intercloud still is in its infancy. A pioneering recent work on dependable Intercloud services is RACS (Redundant Array of Cloud Storage) [1], which, roughly speaking, casts traditional RAID data availability techniques to the Intercloud model and offers better protection against lock-in to a single cloud storage vendor. In the next section, we present our design for *Intercloud storage* (*ICStore*) with dependability goals that exceed those of RACS, notably in terms of improving confidentiality, integrity and consistency, but also with respect to the client fault-tolerance.

3 Intercloud Storage (ICStore)

Our high-level design of ICStore is depicted in Figure 2. At its heart is an ICStore client which orchestrates multiple commodity cloud storage services of the Intercloud (e.g., Amazon S3, Eucalyptus/Walrus [15], Nirvanix) and provides a transparent storage service to the end client who does not need to be aware of the ICStore details, such as the number of different clouds and their APIs. The ICStore client offers to the end client a key-value store with simple read and write operations, which is a common base service offered by commodity cloud storage providers. Our initial implementation views this ICStore/client interface as a subset of the Amazon S3 interface (including, e.g., $put(key, version, value)$, $get(key)$ and $delete(key)$ operations). Eventually, we plan to replace this interface with a standard "simple" storage API, e.g., the SimpleCloud API[5]. With this design, the ICStore client appears to the end client as a single virtual cloud, which simplifies the porting of existing cloud storage applications (e.g., those built on top of Amazon S3) to ICStore. Multiple end clients can access ICStore through its own ICStore client.

At the other end, i.e., in the Intercloud back-end, the ICStore client connects to separate commodity storage vendors, shielding the end client from their API diversity. New cloud storage vendors can be added modularly by writing adaptors for individual vendor APIs to

[5]`http://www.simplecloud.org/`

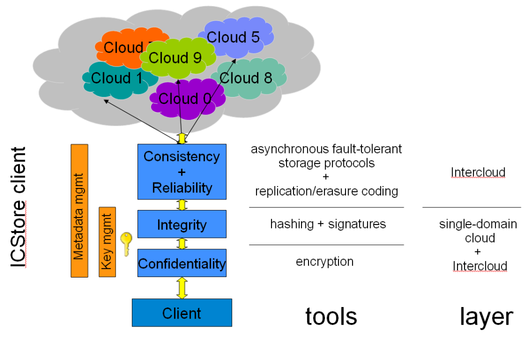

Figure 2: High level design of Intercloud storage.

the more standardized API that follows the one the ICStore exposes to the end client.

Finally, ICStore client consists of three core *layers* that target different dependability aspects: i) confidentiality, ii) integrity and iii) reliability and consistency (RC). This layered approach allows individual layers to be switched "on" and "off" to provide different levels of dependability that are to be matched with client's goals, also with performance and possibly even monetary constraints in mind. In addition, each layer can be individually tuned, as we detail in the following. Note that the ICStore client functionality does not need to be implemented on the client-side. Namely, ICStore client could reside remotely, acting as a separate service and as a gateway to the Intercloud. However, in our initial prototype, ICStore clients are implemented as a library on the end clients.

Confidentiality. In this layer, the client performs a simple symmetric key encryption of the data (i.e., blob value) received from the client. The critical challenge in this layer is key management. To this end, our design supports the use of enterprise-level key managers [4] assuming that such managers are properly replicated to provide a desired level of fault-tolerance.[6]

However, ICStore also supports "in-band" key management, suitable for a fault-prone client scenario we discussed in Section 2.1. Here, a key is split (i.e., secret shared [17]) upon encryption, and key shares are attached as metadata to the pieces of data destined to individual clouds. The number of shares needed to reconstruct the key is a parameter that depends on the number of available clouds and desired resilience and is related to the reliability protocols used in the RC layer. Here we make a rather realistic assumption that different cloud providers do not, in principle, collude to violate data confidentiality. This approach demonstrates how confidentiality issues in a single-domain cloud (mentioned before) can be mitigated by distributing trust in the Intercloud.

[6]Note that this approach to key management is also applicable to a single-domain cloud.

Our approach here is to project the (theoretical) solution of [9] to the (Inter)cloud. The challenge in adapting techniques of [9] to fit our needs lies in the fact that end storage "nodes" (clouds) cannot perform arbitrary computations on data, because their APIs do not support this.[7] Moreover, commodity cloud storage "nodes" cannot initiate communication among each other.

Integrity. The integrity layer provides cryptographic protection against unauthorized data modification. When only a single client accesses the untrusted cloud storage, data integrity can basically be realized with hash trees. For allowing multiple readers and writers to the stored data, the integrity layer relies on a public-key infrastructure at the clients. They issue signatures on the roots of hash trees over the data and exchange those via the untrusted cloud. No extra client-to-client communication needs to be introduced for this purpose.

When multiple clients concurrently access some data maintained by ICstore, a storage-integrity protocol based on fork-linearizability [14] that realizes fail-aware storage [6] is employed. These methods guarantee that data in the cloud cannot be modified without the clients detecting this, even when only one, single-provider cloud is present. Moreover, in the Intercloud, these methods can leverage correct clouds to mask faulty ones and offer full data integrity beyond fork-linearizability.

Reliability and Consistency (RC). The RC layer consists of fault-tolerant distributed protocols that disperse data to the Intercloud after the data (optionally) passes through the confidentiality and integrity layers. In the RC layer, we plan to support a variety of data dispersal protocols which are to be selected depending on the goals of the end application, again respecting performance and monetary constraints.

The simplest reliability-enhancing distributed protocol consists of data replication across multiple clouds. However, as this approach may be prohibitively costly we also envision support for erasure coding techniques (e.g., RAID 5) in the vein of RACS [1] or HAIL [5]. However, we believe that casting techniques such as RAID into cloud may lead to vulnerabilities because these techniques were designed with a synchronous communication model in mind, along with the RAID controller as a single point of failure. In contrast, cloud and Intercloud storage involves communication among WANs, which may lead to (transient) network availability issues as discussed. Moreover, we believe that supporting fault-prone client (proxies) is critical in the Intercloud. To this end, we extend the approach of RACS by supporting asynchronous communication and fault-prone client proxies by applying fault-tolerant asynchronous distributed storage protocols to boost reliability; such protocols can be seen as a generalization of RAID protocols (see e.g., [8] for an overview). In this context and because of limitations in the computational and communication capabilities of cloud storage "nodes," our first ICStore prototype focuses on (variants of) client-driven protocols (e.g., [3]) rather than on more sophisticated protocols that require a lot of computation on storage nodes and mutual communication among them (e.g., [11]).

Furthermore, such asynchronous storage protocols come with different consistency guarantees and a variety of fault-tolerance guarantees (e.g., crash or arbitrary/Byzantine faults). In addition to improving reliability, we plan to boost consistency of Intercloud storage by leveraging fault-tolerance techniques; for example, an inconsistent reply coming from one cloud can be masked by waiting for a sufficient number of consistent replies, or even by declaring an inconsistent (i.e., a stale) reply as Byzantine and leverage Byzantine fault-tolerance techniques to mask such a reply. Clearly, there is a (very small) possibility that all replies from eventually consistent "base" clouds are inconsistent, in which case the ICStore client cannot detect such replies are stale. In such cases, ICStore gracefully degrades to eventual consistency.

Our preliminary experiments indicate that employing a strongly consistent, atomic storage emulation (e.g., [3]) might simply be an overkill for ICStore, especially because such protocols require readers to write back some portion of data to storage nodes. In addition to the increased cost of such an approach, this raises issues regarding the proper configuration of access control on base clouds. Instead, simpler and more lightweight protocols that aim at regular [13] consistency are more suitable. This trend is also present in recent single-domain cloud storage implementations, e.g, [2], which are designed to provide regular (eventual) consistency for a single cloud. Finally, in the later stages of our implementation, we plan to add support for erasure coding to our client-driven storage protocols.

4 Related Work

For lack of space, we omit a comprehensive comparison of our ICStore with all relevant storage systems and focus on the work that is most directly related.

Many commercial cloud storage providers offer geographical diversity (e.g., Amazon S3, Cleversafe[8], EMC

[7]Here we initially aim to avoid bringing up a virtual machine (in e.g., Amazon EC2 or Eucalyptus) to perform the computation, because of the related cost and management overhead. Instead we want to rely on pure commodity storage clouds in the Intercloud back-end. However, exploiting VMs to perform computation on data in the Intercloud remains a viable path to explore.

[8]`http://www.cleversafe.com/`

Atmos[9], Nirvanix, Wuala[10]), either by employing full replication or erasure coding. This approach obviously contributes to better failure independence and dependability, but falls short of the failure independence in the Intercloud, which also provides different implementations and administrative/trust domains. Some providers, e.g., Wuala, also encrypt data on the client side and employ "out-of-band" key management on the client-side [10] to ensure data confidentiality on the cloud. In contrast, we propose an "in-band" key management in ICStore in which keys are split among cloud providers with shares attached as metadata to data (chunks).

HAIL [5] uses erasure coding to disperse data over multiple providers, of which a fraction may collude against the user. It combines cryptographically sound proofs for the retrievability of the data (so that the provider cannot only pretend to have stored it) with the erasure-coded distributed storage. For data integrity, HAIL relies on a symmetric-key MAC, which the users must keep secret. The retrievability methods of HAIL may be combined with our ICStore architecture, but our integrity guarantees are stronger.

Our approach is closest to that of RACS [1], which casts RAID techniques to the Intercloud, as we have already discussed. However, ICStore goes beyond RACS in dependability guarantees by addressing confidentiality, integrity and consistency and also allows for client failures and asynchrony by employing asynchronous fault-tolerant client-driven storage protocols [8].

5 Conclusion

Intercloud is a promising second layer in the dependable cloud computing stack. We presented the design for Intercloud storage (ICStore), which aims to address dependability issues in cloud computing, such as confidentiality, integrity, reliability and consistency. In the near future, we plan to complete the implementation of our ICStore prototype and evaluate its costs and benefits.

References

[1] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: A case for cloud storage diversity," in *ACM SoCC*, 2010. To appear.

[2] E. Anderson, X. Li, A. Merchant, M. A. Shah, K. Smathers, J. Tucek, M. Uysal, and J. J. Wylie, "Efficient eventual consistency in Pahoehoe, an erasure-coded key-blob archive," in *DSN*, 2010. To appear.

[9] `http://www.emc.com/products/detail/software/atmos.htm`

[10] `http://www.wuala.com/`

[3] H. Attiya, A. Bar-Noy, and D. Dolev, "Sharing memory robustly in message-passing systems," *J. ACM*, vol. 42, no. 1, pp. 124–142, 1995.

[4] M. Björkqvist, C. Cachin, R. Haas, X.-Y. Hu, A. Kurmus, R. Pawlitzek, and M. Vukolić, "Design and implementation of a key-lifecycle management system," in *Financial Cryptography and Data Security (FC 2010)*, pp. 160–174, 2010.

[5] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," in *ACM CCS*, pp. 187–198, 2009.

[6] C. Cachin, I. Keidar, and A. Shraer, "Fail-aware untrusted storage," in *DSN*, pp. 494–503, June 2009.

[7] C. Cachin, I. Keidar, and A. Shraer, "Trusting the cloud," *SIGACT News*, vol. 40, no. 2, pp. 81–86, 2009.

[8] G. Chockler, R. Guerraoui, I. Keidar, and M. Vukolić, "Reliable distributed storage," *IEEE Computer*, vol. 42, no. 4, pp. 60–67, 2009.

[9] J. A. Garay, R. Gennaro, C. S. Jutla, and T. Rabin, "Secure distributed storage and retrieval," *Theor. Comput. Sci.*, vol. 243, no. 1-2, pp. 363–389, 2000.

[10] D. Grolimund, L. Meisser, S. Schmid, and R. Wattenhofer, "Cryptree: A folder tree structure for cryptographic file systems," in *SRDS*, pp. 189–198, 2006.

[11] J. Hendricks, G. R. Ganger, and M. K. Reiter, "Low-overhead byzantine fault-tolerant storage," in *SOSP*, pp. 73–86, 2007.

[12] F. Junqueira and K. Marzullo, "A framework for the design of dependent-failure algorithms," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 17, pp. 2255–2269, 2007.

[13] L. Lamport, "On interprocess communication," *Distributed Computing*, vol. 1, pp. 77–101, May 1986.

[14] D. Mazières and D. Shasha, "Building secure file systems out of Byzantine storage," in *PODC*, 2002.

[15] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in *CCGRID*, pp. 124–131, 2009.

[16] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *HotCloud*, 2009.

[17] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[18] W. Vogels, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, pp. 40–44, 2009.