An Untold Story of Redundant Clouds: Making Your Service Deployment Truly Reliable

Ennan Zhai[†] Ruichuan Chen[‡] David Isaac Wolinsky[†] Bryan Ford[†]
[†]Yale University [‡]Bell Labs / Alcatel-Lucent

ABSTRACT

To enhance the reliability of cloud services, many application providers leverage multiple cloud providers for redundancy. Unfortunately, such techniques fail to recognize that seemingly independent redundant clouds may share third-party infrastructure components, e.g., power sources and Internet routers, which could potentially undermine this redundancy.

This paper presents *iRec*, a cloud independence recommender system. iRec recommends at best-effort independent redundancy services to application providers based on their requirements, minimizing costly and ineffective redundancy deployments. At iRec's heart lies a novel protocol that calculates the weighted number of overlapping infrastructure components among different cloud providers, while preserving the secrecy of each cloud provider's proprietary information. We sketch the iRec design, and discuss challenges and practical issues.

1. INTRODUCTION

Lured by properties such as elastic provisioning and scaling, companies offering online applications and services are increasingly moving their computations and data away from private infrastructure into the cloud. While application providers enjoy the benefits of the cloud paradigm, the fact that the cloud abstraction obscures what happens inside remains a significant concern [11, 19, 23, 24]. Many application providers choose to rent from multiple cloud services for redundancy, thereby enhancing the reliability of their applications. Netflix, for instance, utilizes three independent Amazon EC2 availability zones redundantly [2], and iCloud builds on both Microsoft Azure and Amazon EC2 for redundancy [1]. Inter-cloud systems [4, 5, 7], which deploy application services redundantly on diverse providers, have been proposed to make applications more dependable.

Unbeknownst to the application providers, however, third-party infrastructure components shared by redundant deployments may undermine reliability enhancement efforts [11, 19]. Redundant cloud services that appear independent may in fact share deep, hidden interdependencies. This interdependence may lead to unexpected correlated failures across the clouds, in turn making applications unavailable when a shared dependency fails.

Suppose an application provider, A, replicates critical state across two different cloud infrastructure providers, B and C, to enhance the application's reliability. A, however, does not know B and C share a third-party infrastructure component, a power source P. If the power source P crashes for whatever reason, both B and C may become simultaneously unavailable, resulting in a correlated failure causing a service outage in application A. This scenario is not merely hypothetical. In one recent report [10], a storm in Dublin, Ireland took out local power source and its backup generator, thus causing both Amazon and Microsoft clouds in that region to be unavailable for many hours, potentially undermining the reliability of applications attempting to achieve redundancy across those two cloud infrastructure providers.

Due to cloud services' non-transparent, proprietary layering of services and infrastructures, it is difficult for application providers to locate the root causes of correlated failures even after they encounter an outage, let alone to anticipate the possible sources of such failures in advance. For cloud providers, some recent efforts have been focused on automatic failure localization and tolerance [3, 6, 9, 15, 18], making their services more robust to failures. Resolving such correlated failures, however, still requires significant human intervention, leading to prolonged failure recovery time [12, 35]. Moreover, cloud providers typically do not know whether they are sharing third-party infrastructure components with other providers [11, 19].

Our goal. Facing this dilemma, this paper explores a new approach. Rather than localizing or tolerating failures, we aim at recommending *truly independent* redundancy services to application providers when they first attempt to deploy redundant systems. This approach minimizes potential correlation between different cloud providers on which the application providers plan to deploy their applications. With this approach, application providers' deployments can be enhanced significantly without wasting resources on ineffective redundancy. To the best of our knowledge, there is no existing effort addressing this problem.

Road-Map. The rest of this paper is organized as follows. We first describe the target scenario and challenges in the next section. We then propose a practical recommender and present its design in Section 3. Important practical issues and challenges are discussed in Section 4. We then summarize related work in Section 5, and outline future work in Section 6.

HotDep - Workshop on Hot Topics in Dependable Systems
November 3, 2013, Nemacolin Woodlands Resort, PA, USA
ACM 978-1-4503-2457-1.

2. SCENARIO & CHALLENGES

Before describing our proposal, we first clarify our target scenario, then discuss key technical challenges.

2.1 Scenario

Figure 1 shows a typical scenario for our target problem. There are three different entities: 1) The application provider, who intends to deploy her application across multiple cloud providers for redundancy. An application provider may be a large organization with vast resources and global interests, or an individual with limited resources and regional interests. Well-known application providers illustrating this role include iCloud, Netflix, and Zynga. 2) Cloud providers, who host infrastructure and services such as storage, computation, and network bandwidth, and may share some third-party infrastructure components, such as power sources and Internet routers. Representative cloud providers include Amazon's EC2 and S3, as well as Microsoft's Azure. 3) The cloud recommender, our proposed system, fills a gap between the application provider and cloud providers. An application provider expresses her deployment expectations and requirements to the cloud recommender, who recommends redundancy-optimized combinations of cloud providers to the application provider.

2.2 Challenges

Designing a practical cloud recommender presents many technical challenges; we summarize two key challenges here.

Challenge 1: Personalized requirements on heterogeneous infrastructure components. Different cloud providers may share various third-party infrastructure components such as power sources, Internet routers, and physical links. Different types of infrastructure components usually play different roles for the cloud providers, have different reliability properties, and are managed differently. For instance, the outage of one power source might affect multiple data centers, but the failures of several core routers may affect only one data center's network connectivity. Cloud providers in principle have access to such infrastructure information, but they normally do not know exactly how their customers, i.e., application providers, will use their services in applications. By contrast, each application provider knows its own dependencies on various types of infrastructure components, though normally only qualitatively. Each application developer, however, is unlikely to have access to detailed information about the structure and interdependence of the infrastructure components underlying the clouds on which the application might build.

As a result, to provide meaningful recommendation results, a cloud recommender must not only consider the importance and structural roles of each type of infrastructure component from a cloud provider's point of view, but also understand each application provider's personalized qualitative requirements. These customer-specific qualitative requirements must then be transformed into quantitative requirements suitable for automated processing.

Challenge 2: Preserving proprietary business secrets. Recommender systems traditionally rely on users' feedback, through voting for example. In our scenario, due to the non-transparency of clouds, it is difficult for application providers to offer feedback after-the-fact on whether their prior redundant configurations have proven adequately independent—except *after* a correlated failure occurs, proving their configuration less redundant than they had hoped. Common dependencies between different cloud providers might be found if, for instance, these cloud providers were willing to share details of their internal infrastructures with each other or with their application provider customers. Such open information sharing by cloud providers seems unlikely to be viable in practice, however, as the internal details of their infrastructures are typically considered closely-guarded secrets critical to their business competitiveness.

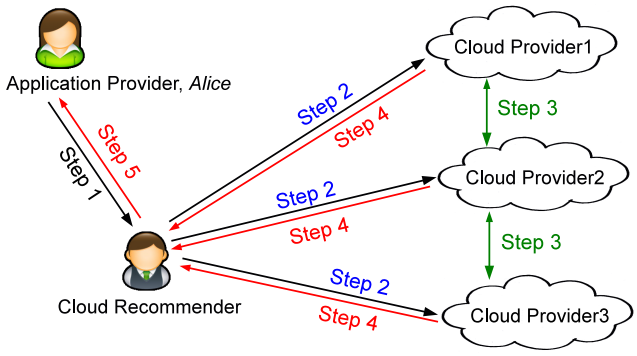

Figure 1: System overview. Step 1: Requirement submission; Step 2: Requirement forwarding; Step 3: W-PSI-CA computation; Step 4: Replying results; and Step 5: Recommendation.

Two obvious solutions would be: 1) to introduce a trusted third party, which collects infrastructure information from multiple cloud providers and computes recommendations for application providers; or 2) using secure multi-party computation (SMPC) [38] to compute recommendation results privately, without revealing the datasets from which those results are computed. Unfortunately, in the former case cloud providers may be hesitant to trust a third party, while the latter option is complex and time-consuming [36,39]. Determining common infrastructure components while preserving the secrecy of cloud providers' proprietary information, therefore, presents a difficult practical challenge.

3. A PRACTICAL RECOMMENDER: iRec

We now propose a recommender system, iRec, designed to recommend redundant cloud configurations while addressing the challenges above. Before presenting iRec's design in detail, we first discuss background the design builds on.

3.1 Background: PSI-CA

Our approach relies on an existing primitive called *private set-intersection cardinality* (PSI-CA) [13, 16]. This algorithm allows a group of k parties with multisets $S_1, ..., S_k$ to compute privately $|\cap_i S_i|$, the number of elements they have in common, without learning the specific elements in $\cap_i S_i$. PSI-CA uses an additive homomorphic encryption scheme such as the Paillier cryptosystem [21].

Suppose we have two parties p_1 and p_2, each having a data set: $S_1 = \{x_1, \cdots, x_m\}$ and $S_2 = \{y_1, \cdots, y_n\}$, respectively. We briefly summarize the PSI-CA algorithm.

- **Step 1:** Party p_1 first generates a polynomial, $P(z)$, over a finite field whose roots are S_1's elements x_i.

Thus, the polynomial generated by p_1 is: $P(z) := (x_1 - z)(x_2 - z)\cdots(x_m - z) = \sum_u \alpha_u z^u$.

- **Step 2:** p_1 sends to p_2 the homomorphically encrypted coefficients α_u of $P(z)$, along with p_1's public key.

- **Step 3:** For each of S_2's elements y_i, p_2 computes $\text{Enc}(rP(y_i) + 0^+)$, evaluating the polynomial at each element in S_2, multiplying each result by a fresh random number r, and finally appending 0^+, a string of zeros. p_2 can compute this without knowing p_1's private key due to the cryptosystem's homomorphism.

- **Step 4:** p_2 sends a random permutation of the results back to p_1, who decrypts them and counts the occurrences of the 0^+, yielding $|S_1 \cap S_2|$.

Though we focus above on a case for computing the cardinality of two set intersection, the primitive extends to deal with the case of multiset intersection [13, 16]. We omit the description of this extension for space reasons.

3.2 The Design of iRec

iRec makes two key assumptions. First, we assume all the entities including application providers, iRec, and cloud providers faithfully follow the specified protocol. Second, each cloud provider knows only the information about its own infrastructure components, and this information is accessible only to the cloud provider itself.

In practice, cloud providers have infrastructure management and monitoring tools to collect and record information about their own and third-party infrastructure components they depend on; thus, we expect they could repurpose this information for use by iRec.

We now outline the heart of iRec's design, deferring to later sections discussion of important details such as unifying the information contexts of different cloud providers.

Step 1: Requirement submission. Alice, an application provider, wants to deploy her application across multiple cloud providers for redundancy. She contacts iRec and conveys her deployment requirements, which typically include: 1) on which redundant clouds Alice might deploy her application; 2) which alternative services she may use; and 3) what types of common infrastructure components presenting correlated failure risks, such as power sources and Internet connectivity, Alice is most concerned about. In addition, Alice also specifies the *weights* of different types of infrastructure components, to obtain results tailored to her application. Weights in our design are positive integers indicating relative importance of different infrastructure components. In practice, if Alice does not have the information needed to tune these weights herself, she may consult the iRec service, choosing some standardized weight assignment scheme.

Step 2: Requirement forwarding. On receiving Alice's requirements, the iRec sends each of Alice's potential cloud providers two lists: 1) the weights Alice assigned to the various infrastructure components, and 2) the list of other alternative cloud providers she is considering.

Step 3: W-PSI-CA computation. With the two lists above, each of Alice's specified cloud providers performs a protocol we call W-PSI-CA (Weighted PSI-CA). Figure 2 illustrates how W-PSI-CA extends PSI-CA to support weights.

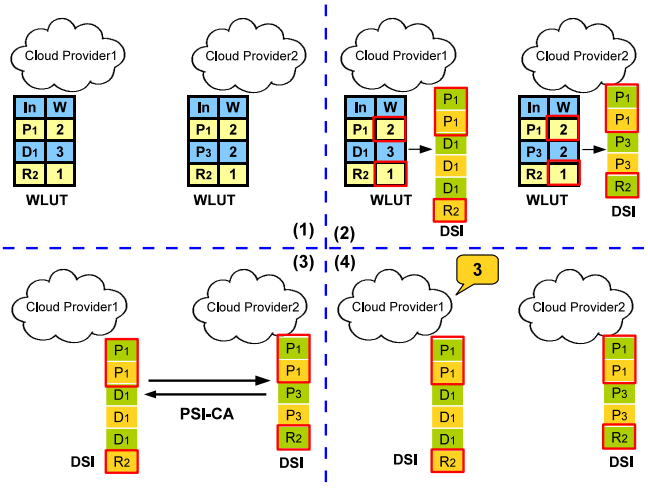

Figure 2: A W-PSI-CA computation. The first column in WLUT gives names of infrastructure components, and the second column gives their weights. P_i denotes power source i, R_j means Internet router j, and D_k is used to denote DNS k. We assume the third-party infrastructure components have uniform identities in this scenario.

Each cloud provider first generates a Weight Look Up Table (WLUT). This table captures the cloud provider's infrastructure components and their corresponding weights, as shown in Figure 2(1). For instance, Cloud Provider1 uses three third-party infrastructure components: power source 1 (P_1), DNS 1 (D_1) and Internet router 2 (R_2). Their weights are 2, 3 and 1, respectively.

Next, the cloud provider generates a Data Set for Intersection (DSI) based on the WLUT. As shown in Figure 2(2), the DSI has only one column. The count of each infrastructure component in the DSI equals the infrastructure component's weight in the WLUT. For example, in Figure 2, since an infrastructure component P_1's weight is 2, two instances of P_1s are present in the DSI.

Each cloud provider then performs the PSI-CA protocol with each other provider's DSI, to obtain the weighted number of shared infrastructure components between different cloud providers. Each cloud provider stores these set intersection cardinalities. Each cloud provider moreover needs to get the sizes of the DSIs of the other cloud providers involved in the PSI-CA protocol. In Figure 2, for example, since two cloud providers share the same power source P_1 and the same routing infrastructure component R_2, Cloud Provider1 computes the set intersection cardinality with Cloud Provider2. The computation result is 3. Cloud Provider1 learns two values: 3, the weighted number of overlapping infrastructure components, and 5, the size of the DSI of Cloud Provider2.

Upon obtaining the above two values, a cloud provider (say i) computes an *overlap rate*, $OR_{i,j}$, with respect to a cloud provider j, using the following equation (1):

$$OR_{i,j} = \frac{WN}{|DSI_i| + |DSI_j| - WN} \quad (1)$$

In this equation, WN is the weighted number of shared infrastructure components between cloud provider i and j, and $|DSI_i|$ is the size of the DSI of cloud provider i. In Fig-

ure 2, for example, Cloud Provider1 calculates the overlap rate as: $3/(6+5-3) = 3/8$.

At this point we have described only the two-party W-PSI-CA. W-PSI-CA is easily extended to the multi-party scenario using generalizations in the original PSI-CA algorithm [13]. Not surprisingly, W-PSI-CA inherits the original PSI-CA's property of ensuring the privacy of each cloud provider's infrastructure information.

Step 4: Collecting results. After the W-PSI-CA, each cloud provider sends all the overlap rates it computed with other providers back to the cloud recommender iRec.

Step 5: Recommendation. With the overlap rates in hand, the iRec recommends Alice the most independent cloud services. Specifically, the iRec ranks the potential cloud provider configurations in order based on the overlap rates, and sends this ordered list to Alice. Since Alice receives only an ordered list reflecting possible configurations, she obtains no proprietary information about the cloud providers' internal infrastructures in this result.

3.3 Differential Privacy Enhancement

Both the original and our weighted PSI-CA primitives privacy-protect each cloud provider's proprietary information. Each cloud provider, however, still learns the *exact* sizes of the resulting intersection and other cloud providers' DSIs. In some cases, even this leakage might be a concern.

An existing effort, DJoin [20], proposes a differentially private form of private set intersection cardinality, BN-PSI-CA. This extension adds differentially private noise to the multi-party set intersection cardinality results, strengthening the offered privacy guarantee. We can thus directly benefit from BN-PSI-CA by replacing PSI-CA with BN-PSI-CA in our W-PSI-CA protocol.

3.4 Example Use Case

We now describe in more detail a hypothetical use case for iRec, inspired by the Netflix situation mentioned in the introduction. Alice, a video-on-demand entrepreneur, wishes to deploy her new video delivery application across three redundant cloud providers, but wants their data centers to be located in the same geographic region to minimize impact on performance. Nevertheless, given that the data centers are physically nearby, Alice is legitimately worried that they may depend on similar local power distribution sources in the area, and may depend on Internet feeds arriving over common fibre optic paths from similar or identical nearby Internet Exchange Points (IXPs).

Alice initially has a choice of six alternative cloud providers who offer suitable services and have data centers in her desired geographical region, and she would like to choose three of them. She finds first that only four of these cloud providers support iRec. She therefore immediately narrows her choices to those four providers that support iRec, as she sees this as a mark of transparency and trustworthiness in cloud provider operation, and without the information provided by iRec she would doubt the true level of redundancy she is actually achieving in her application. In effect, supporting iRec may give cloud providers a way to bolster their "reliability credentials" and distinguish themselves from their competitors, to their own advantage.

The cloud providers that support iRec are required by the iRec standardization to include certain tags in their WLUTs and DSIs, representing the specific local power sources, power cables, IXPs and fibre optic cables their data centers depend on. In this way, through their contractual commitments with the iRec standardization body, each cloud provider who depends on the same power sources and Internet supply components will use the same standardized WLUT and DSI tags in describing those dependencies. Through this standardization, iRec identifies and measures the relative importance of any overlapping infrastructure components on Alice's behalf, through the W-PSI-CA computation above.

4. PRACTICAL ISSUES

This section explores potential issues for iRec deployment.

4.1 Do providers have incentives to join?

The motivation for application providers to use iRec is straightforward: they can choose redundant cloud services offering better independence in combination, improving reliability and avoiding the costs of ineffective redundancy. In contrast, cloud providers may not benefit directly from participating in such a recommendation process. Therefore, what incentive do cloud providers have to join iRec?

One possible benefit to cloud providers is that participating in iRec may enable providers to understand their potential infrastructure issues better in relation to other providers. While cloud providers may not learn which specific infrastructure components overlap with others, they can learn to what extent common dependencies exist between different cloud providers. iRec thus gives cloud providers as well as their customers the opportunity to measure and improve the independence of their infrastructure deployments, in order to head off potential reliability risks.

Another possible incentive is that cloud providers not participating in iRec will not appear among the alternative cloud providers iRec offers application providers. Application providers may be less likely to learn about or consider these alternatives when they evaluate possible redundant application deployments. Thus, non-participating cloud providers might lose potential customers due either to the lack of the iRec "reliability label" or merely due to not being on the iRec "certified provider list."

Finally, iRec offers cloud providers the opportunity to improve their reputation for reliability and transparency, without risking significant leaks of proprietary secrets about the structure of their cloud infrastructure. Thus, joining iRec offers cloud providers a privacy-preserving way to increase the effective transparency of their infrastructures.

4.2 Will cloud providers behave honestly?

In reality, some cloud providers execute the iRec protocol dishonestly. They might "declare" only a subset of their true infrastructure component dependencies in their WLUTs, for example, thereby generating partial DSIs while performing W-PSI-CA. In doing so, these cloud providers might benefit from their dishonesty by appearing to have a smaller set intersection cardinality and hence greater independence from other providers. Dishonest cloud providers may thus rank higher in the resulting recommendation lists.

One approach to address this issue might be to use trusted hardware (e.g., TPM) to remotely attest whether cloud providers are performing the W-PSI-CA as required. Since existing efforts such as Excalibur [23] have deployed TPM into cloud services, the TPM could in principle attest to the

behavior of a cloud provider, verifying whether the cloud provider honestly computed the set intersection.

The inputs into the W-PSI-CA computation would also have to be TPM-protected, however, to prevent a dishonest provider from doctoring the inputs before they enter the cooperative computation. Producing trusted inputs to W-PSI-CA may thus imply incorporating TPM into potentially shared components such as upstream Internet routers or power sources, a task we expect is possible in principle though extremely challenging in practice.

A more practical alternative might be to introduce a cloud auditor, who most of the time does not have access to the inner structural workings of individual cloud providers, but who has the authority to make occasional spot-check "inspections", either at random times or in response to a complaint by an application provider. Other approaches to cloud reliability auditing already assume a trusted third-party auditor [39]. iRec might complement such approaches by reducing the auditor's load and day-to-day information access requirements, while ensuring that cloud providers risk exposure if they implement iRec dishonestly.

4.3 Is an auditing system an alternative?

As suggested by Shah et al. [25], cloud auditing systems may be capable of discovering shared infrastructure components across multiple cloud providers. A cloud auditing system might build a dependency graph to represent dependencies between the infrastructure components of alternative cloud providers. Based on this dependency graph, the auditing system can locate common infrastructure components across clouds. A natural question is why we do not directly adopt cloud auditing system to achieve our goal of recommending independent redundant services.

While this more straightforward design would work, we suspect cloud providers might be more resistant to adopting it because of the greater information disclosure and trust they must place in the third-party auditor. Traditional cloud auditing systems need full dependency information about all relevant components within each cloud, and must centrally aggregate all cloud providers' sensitive infrastructure information to generate the entire dependency graph. Cloud providers may be unwilling to share this full internal dependency information on a regular basis.

While some privacy preserving auditing protocols for the integrity of cloud storage services have been proposed [26, 30–33, 37], these only enable application providers to check whether their data has been maliciously modified, and do not offer any guarantee that the application service will remain reliable and available.

Another alternative solution would make use of secure multi-party computation (SMPC) to avoid privacy leakage. In our prior work [36, 39], we used this method to find common components across cloud providers. While more general, these types of SMPC unfortunately scale poorly.

5. RELATED WORK

There is surprisingly little existing work on the design of cloud independence recommendation systems. To the best of our knowledge, iRec is the first effort in this space.

Nevertheless, existing efforts have focused on determing suitable cloud providers to meet other user-specified requirements, such as finding the right balance between performance and cost. CloudCmp [17] developed a benchmark tool for computing the cost-to-performance ratio of four representative cloud providers. Other studies [28, 29] have made similar efforts to deploy applications in consideration of other cloud characteristics and objectives.

CloudWard [14] explored the problem of deploying component-based applications in the cloud, formulating this challenge as an optimization problem by maximizing deployment benefit while satisfying client-imposed policy constraints. Similarly, Conductor [34] focuses on MapReduce-type workloads, and KingFisher [27] proposes cost-aware provisioning in clouds. These three efforts might be viewed as cloud recommender systems for different purposes. None of this work evaluates infrastructure independence or recommends cloud deployments to maximize the benefits of redundancy.

In a prior technical report we proposed Structural Reliability Auditors (SRA) [39], the first architecture we know of for systematically discovering common dependencies and correlated failure risks in cloud infrastructure. Building on this idea, Xiao et al propose a privacy-enhanced SRA, P-SRA [36], which determines potential common dependencies using Secure Multi-Party Computation (SMPC) techniques. While more general, SPMC is extremely time-consuming, and does not scale well to cloud infrastructure datasets in which there may be a large number of common infrastructure components. The iRec design in contrast focuses on developing a simple, best-effort recommendation scheme to measure the independence of cloud services for redundancy purposes, while preserving cloud providers' information using the more efficient and scalable PSI-CA algorithm.

6. CONCLUSION AND FUTURE WORK

To address the risk of correlated failures resulting from common third-party infrastructure components shared by multiple cloud providers, we have sketched a privacy-preserving scheme for evaluating the independence of alternative cloud services, and recommending combinations to application providers maximizing the benefits of redundancy.

For future work, we intend to improve iRec in the following ways: 1) To make iRec more practical, we will explore the design space of more effective weight assignment schemes. We intend to adapt traditional analytic hierarchy process techniques [22] to reason about iRec's weight assignment process. This work may not be trivially applicable to iRec, since there are many factors that may result in correlated cloud failures. 2) We are developing an iRec prototype and plan to evaluate its effectiveness and efficiency using realistic datasets. 3) A high-performance differential privacy mechanism has been proposed in SplitX [8]; we are planning to enhance the efficiency of differential privacy in iRec via a similar approach. 4) To ensure the correctness of computed results, we would like to explore incentive schemes that might encourage cloud providers to follow the W-PSI-CA protocol honestly. 5) Handling multi-layer third-party infrastructure components is an important future problem, since real clouds can be complex and many-layered.

Since we believe cloud provider independence to be of real interest to many application providers, further research along these lines may yield a variety of new cloud-based independence measurement and recommendation techniques, not only related to reliability.

Finally, iRec suggests potential opportunities to optimize other goals. While the emphasis of this paper is on reliabil-

ity, we envision iRec could evolve into or be combined with a more comprehensive cloud recommender system.

Acknowledgments

We wish to thank the anonymous HotDep reviewers for their comments. We also thank Jeff Mogul and Gustavo Alonso for helpful suggestions. This work is supported by the National Science Foundation under Grant No. CNS-1149936.

7. REFERENCES

[1] Apple's iCloud runs on Microsoft's Azure and Amazon's cloud. `http://venturebeat.com/2011/09/03/icloud-azure-amazon/`.

[2] Netflix. `https://signup.netflix.com/`.

[3] P. Bahl, R. Chandra, A. G. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards Highly Reliable Enterprise Network Services Via Inference of Multi-level Dependencies. In *SIGCOMM*, pages 13–24, 2007.

[4] C. Basescu, C. Cachin, I. Eyal, R. Haas, A. Sorniotti, M. Vukolic, and I. Zachevsky. Robust Data Sharing with Key-Value Stores. In *DSN*, pages 1–12, 2012.

[5] A. N. Bessani, M. P. Correia, B. Quaresma, F. André, and P. Sousa. DepSky: Dependable and Secure Storage in a Cloud-of-clouds. In *EuroSys*, pages 31–46, 2011.

[6] N. Bonvin, T. G. Papaioannou, and K. Aberer. A Self-organized, Fault-tolerant and Scalable Replication Scheme for Cloud Storage. In *SoCC*, 2010.

[7] C. Cachin, R. Haas, and M. Vukolic. Dependable Storage in the Intercloud. Technical Report RZ3783, IBM Research, Zurich, 2010. Available at `http://domino.research.ibm.com/library/cyberdig.nsf/papers/630549C46339936C852577C200291E78/$File/rz3783.pdf`.

[8] R. Chen, I. E. Akkus, and P. Francis. SplitX: High-Performance Private Analytics. In *SIGCOMM*, pages 315–326, 2013.

[9] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions. In *OSDI*, pages 117–130, 2008.

[10] J. Clark. Lightning strikes Amazon's European cloud. `http://www.zdnet.com/lightning-strikes-amazons-european-cloud-3040093641/`.

[11] B. Ford. Icebergs in the Clouds: the Other Risks of Cloud Computing. In *HotCloud*, 2012.

[12] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in Globally Distributed Storage Systems. In *OSDI*, pages 61–74, 2010.

[13] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT*, pages 1–19, 2004.

[14] M. Y. Hajjat, X. Sun, Y.-W. E. Sung, D. A. Maltz, S. G. Rao, K. Sripanidkulchai, and M. Tawarmalani. CloudWard Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. In *SIGCOMM*, pages 243–254, 2010.

[15] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed Diagnosis in Enterprise Networks. In *SIGCOMM*, pages 243–254, 2009.

[16] L. Kissner and D. X. Song. Privacy-Preserving Set Operations. In *CRYPTO*, pages 241–257, 2005.

[17] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: Comparing Public Cloud Providers. In *IMC*, pages 1–14, 2010.

[18] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson. F10: A Fault-Tolerant Engineered Network. In *NSDI*, 2013.

[19] J. C. Mogul. Emergent (Mis)behavior vs. Complex Software Systems. In *EuroSys*, pages 293–304, 2006.

[20] A. Narayan and A. Haeberlen. DJoin: Differentially Private Join Queries over Distributed Databases. In *OSDI*, 2012.

[21] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, pages 223–238, 1999.

[22] T. L. Saaty. *What is the analytic hierarchy process?* Springer, 1988.

[23] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu. Policy-Sealed Data: A New Abstraction for Building Trusted Cloud Services. In *USENIX Security*, 2012.

[24] M. Schwarzkopf, D. G. Murray, and S. Hand. The Seven Deadly Sins of Cloud Computing Research. In *HotCloud*, 2012.

[25] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan. Auditing to Keep Online Storage Services Honest. In *HotOS*, 2007.

[26] M. A. Shah, R. Swaminathan, and M. Baker. Privacy-Preserving Audit and Extraction of Digital Contents. *IACR Cryptology ePrint Archive*, 2008:186, 2008.

[27] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A Cost-Aware Elasticity Provisioning System for the Cloud. In *ICDCS*, pages 559–570, 2011.

[28] P. B. Teregowda, B. Urgaonkar, and C. L. Giles. CiteSeer^x: A Cloud Perspective. In *HotCloud*, 2010.

[29] E. Walker. Benchmarking Amazon EC2 for High-Performance Scientific Computing. *USENIX Login*, 33(5):18–23, 2008.

[30] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou. Privacy-Preserving Public Auditing for Secure Cloud Storage. *IEEE Trans. Computers*, 62(2):362–375, 2013.

[31] C. Wang, K. Ren, W. Lou, and J. Li. Toward Publicly Auditable Secure Cloud Data Storage Services. *IEEE Network*, 24(4):19–24, 2010.

[32] C. Wang, Q. Wang, K. Ren, and W. Lou. Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing. In *INFOCOM*, pages 525–533, 2010.

[33] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li. Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing. *IEEE Trans. Parallel Distrib. Syst.*, 22(5):847–859, 2011.

[34] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Orchestrating the Deployment of Computations in the Cloud with Conductor. In *NSDI*, 2012.

[35] X. Wu, D. Turner, C.-C. Chen, D. A. Maltz, X. Yang, L. Yuan, and M. Zhang. NetPilot: Automating Datacenter Network Failure Mitigation. In *SIGCOMM*, pages 419–430, 2012.

[36] H. Xiao, B. Ford, and J. Feigenbaum. Structural Cloud Audits that Protect Private Information. In *CCSW*, 2013.

[37] K. Yang and X. Jia. Data Storage Auditing Service in Cloud Computing: Challenges, Methods and Opportunities. *World Wide Web*, 15(4):409–428, 2012.

[38] A. C.-C. Yao. Protocols for Secure Computations (Extended Abstract). In *FOCS*, pages 160–164, 1982.

[39] E. Zhai, D. I. Wolinsky, H. Xiao, H. Liu, X. Su, and B. Ford. Auditing the Structural Reliability of the Clouds. Technical Report YALEU/DCS/TR-1479, Department of Computer Science, Yale University, 2013. Available at `http://www.cs.yale.edu/homes/zhai-ennan/sra.pdf`.