Overcoming Risks in Hidden Dependencies within the Cloud

Eric Chang
Yale University

Abstract

In order to address reliability and availability of cloud services, redundancy-based techniques utilizing more than one cloud provider have been recently proposed. Unfortunately these approaches fail to recognize the effects related to common dependencies hidden by independent redundant services, potentially invalidating these efforts. We propose a novel system to address this pitfall by recommending cloud consumers the most suitable redundant services based on the consumers' requirements. We call this system a cloud reliability recommender (CRR). At the heart of a CRR, we leverage fault tree analysis techniques to 1) discover hidden common dependencies in order to generate correlation matrices; 2) calculate the failure probabilities of alternative services; and, 3) enable cloud consumers to specify criteria to optimize their utility, recommending the most suitable services for them. In this paper, we describe the CRR, the aforementioned process, and discuss the challenges inherent to CRR design and practicality.

1 Introduction

Background and target problem. Using the cloud or having services in the cloud have become popular business cliches as increasingly more companies move their computations and data away from personally owned resources onto third-party resources called the cloud. Cloud consumers or application service providers enjoy the simplicity that comes with leveraging the cloud, but unfortunately the cloud also serves to obscure what happens inside. These concerns have led into investigations on the availability and reliability of cloud services [5].

Based on such intuition, redundancy-based techniques have been proposed to guarantee the availability and reliability of cloud services [6, 8, 25]. Nevertheless, cloud service characteristics, non-transparent layering of services and infrastructures, potentially hide some pitfalls which might invalidate these availability and reliability enhancement techniques [10, 14, 20, 21, 24]. Concretely, alternative redundant services which appear independent might actually share deep, hidden dependencies which may lead to unexpected correlated failures. An application provider deploying their services into the cloud, or a cloud consumer, might be oblivious to such dependencies when attempting to construct a fault tolerant cloud deployment resulting in costly and ineffective attempts to produce a reliable infrastructure [7, 22].

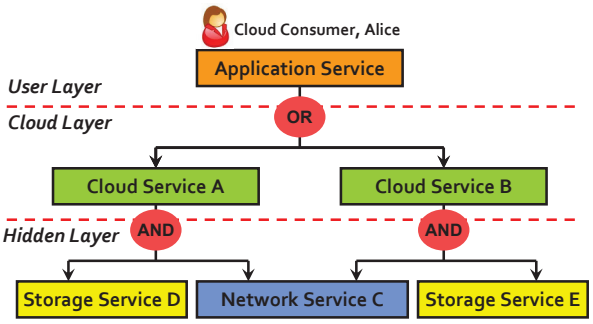

Figure 1: A typical example of illustrating our target problem.

Even now, many cloud consumers apply approaches to overcome failures due to a single data center or cloud crash. Netflix [2], for instance, utilizes three independent Amazon EC2 regions to limit shared dependencies, while Zynga [3], developers of many Facebook games, uses both EC2 and an internal cloud system to enhance its availability [1]. During the EC2 crash in April of 2011, both experienced significant failures, but because the failures were isolated, their redundancy approaches sufficed. While admirable and successful in this circumstance, their approaches fail to address the problem fundamentally, as they are still unaware of "secrets" or deep dependencies hidden by non-transparent cloud services. A recent study [10] shows that this problem has not yet been paid significant attention, potentially creating a dangerous situation waiting to erupt.

Figure 1 illustrates a situation where redundancy techniques may not be sufficient to provide increased reliability. In this example, the cloud consumer, Alice, replicates critical states on both cloud services A and B, for ensuring reliability of the application service. Alice, however, does not know that both A and B share a common dependency: network service C. If C becomes unstable, Alice's service will begin exhibiting unexpected behavior. Alice's unknown shared dependency on C has been hidden by both A and B but presents a hidden correlated failure.

Facing this dilemma, this paper aims to answer the following central question:

Can we build a system that recommends suitable redundant services to cloud consumers according to their requirements (e.g., reliability, cost, and location) while minimizing hidden common dependencies?

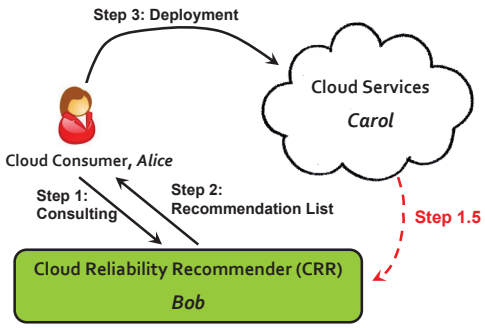

Figure 2: Target scenario and the interacting among the cloud consumer (Alice), cloud services (Carol) and the CRR (Bob).

Building such a system would significantly enhance cloud consumers' selections, thereby allowing them to improve their cloud deployments without wasting resources on unnecessary redundancies, i.e., the ones having correlated failures.

Our approach. Motivated by the above question, this paper proposes a novel approach for recommending the most suitable (redundant) services to cloud consumers in terms of their specifications. In order to facilitate the recommendation service, we construct a cloud reliability recommender (CRR), who translates clients' expectations and requirements into deployment models across various services hosted by a variety of cloud providers. Upon receiving a request from a consumer, the CRR makes a recommendation utilizing both the failure probabilities as calculated by fault tree analysis [17] and correlation matrices representing hidden common dependencies in order to reduce ineffective duplication while emphasizing the consumer's other criteria.

Road-map. The rest of this paper is organized as follows. The next section describes system architecture. Section 3 presents detailed design of our approach, and Section 4 illustrates our system with a case study. Section 5 finally discusses our future work.

2 System Architecture

This section discusses interactions in our system architecture between the cloud reliability recommender (CRR) and the cloud consumer as well as the CRR and the cloud providers.

Target application/scenarios. In our system, there exist three different roles: 1) cloud consumers intending to deploy their services into cloud; 2) cloud service providers hosting various services such as storage, computation and network communications; and, 3) cloud reliability recommender (CRR) recommending suitable (redundant) services to cloud consumers. As shown in Figure 2, it presents a typical scenario we target.

Usage model of our system. In addition to highlighting the roles in our system, Figure 2 also demonstrates one potential usage model. Specifically, a cloud consumer, Alice, wants to deploy her application service to the cloud while ensuring reliability. In Step 1, Alice contacts a cloud reliability recommender (CRR), Bob. Bob discusses with Alice her deployment expectations and potential scenarios resulting in a deployment model tailored for Alice, Step 2. After receiving a recommendation from Bob, Alice adopts her service onto some cloud providers, Step 3. In the intermediate Step 1.5, Bob consults the cloud providers, this could either be done inline or before Alice's inquiry. In this scenario, Bob plays the role of a third party, an ombudsman between Alice, the cloud consumer, and a variety of cloud providers. In the remainder of this section we discuss what the interaction between Alice and Bob followed by the interaction between Bob and the cloud providers.

2.1 Conversing with Cloud Consumers

Cloud consumers may be large organizations with vast resources and global interests or individuals with limited resources and regional interests with a myriad of other possibilities between the two. Alice, a cloud consumer, cannot realistically go to Bob, a CRR, and ask for a set of the most reliable cloud services for her application deployment without at least some restrictions, because Bob may recommend services in some remote land, far away from Alice's target customers, or instead he might recommend very reliable services costing orders of magnitude more than competitors. In order to fine tune the deployment plan Bob provides to Alice, we envision that Alice's communication with Bob will involve a discussion of Alice's goals, such as, locality, cost, importance of reliability, and types of services.

Alice may want to speak directly with a human being; thus, a CRR could actually be a consulting service staffed with real people, such as Bob, who interact with an internal system in order to provide recommendations. This approach provides robustness capable of handling non-technical expectations translating them into a specific set of requirements. From there, Bob could even help Alice deploy her services, allowing non-technical individuals and business make better use of the cloud.

Alternatively, Alice may be very technical wanting to interact with a web service-based CRR. In this scenario, the CRR would have an API consisting of a function that when queried returns back a deployment schema. Alice could take the schema input it into a deployment application, instantaneously deploying her system into the cloud. Alternatively she could read the schema and manually do the process. We envision such an API would have at least one function: `getDeployment(params[])`. This function might

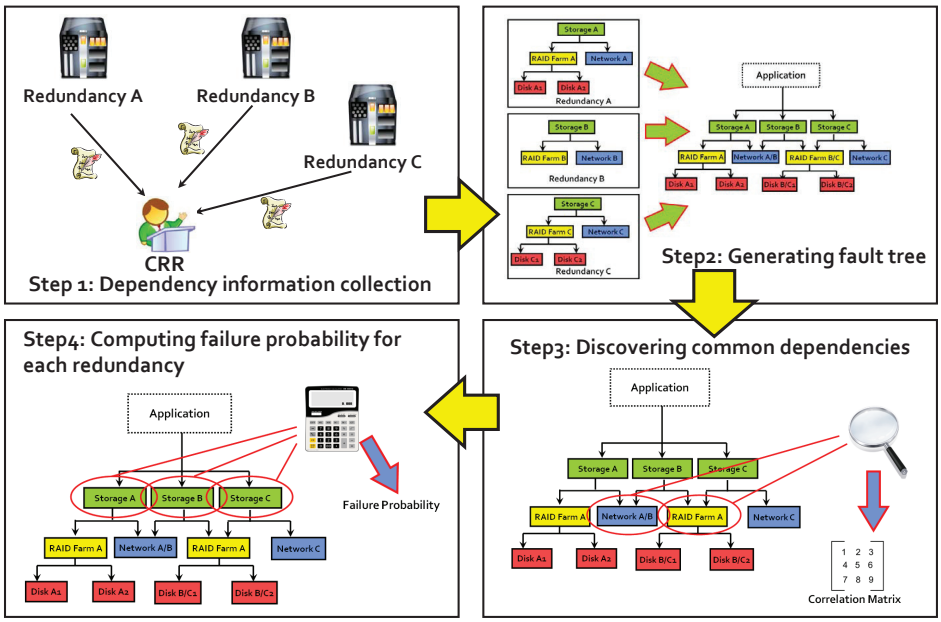

Figure 3: Illustrative examples of four steps in our approach.

have many parameters, such as total cost, cost goals, performance goals, service type, network demands, location, or reliability expectations. Each parameter could have a default value, but Alice could specify the values that are important for her.

2.2 Discovering Dependencies

The construction of clouds are well kept secrets. For example, searching the Internet for details about Amazon or Google's does not reveal much. Asking them directly tends to be an equally unhelpful approach. This presents a difficult challenge for the CRR, as the CRR must understand deep, potentially secret, aspects of a cloud in order to calculate reliability and make recommendations.

A practical approach to allowing cloud providers to share dependency information with a CRR would be through NDA (non-disclosure agreement). With the NDA, a CRR could easily document all the dependencies and be kept up to date as changes occur. We believe that this is practical because it benefits both the consumer and the provider with minimal risk for the provider [16]. By inclusion into the CRR's database of cloud providers, the provider may see a rise in business due to recommendations to consumers from the CRR. Cloud providers may selectively sign NDAs with large consumers but not with small companies. The CRR could represent the dispersed group as a single entity acting as a proxy between the companies secrets and the consumers' goals.

If for some reason, a cloud provider does not want to sign NDAs, the CRR could investigate through other means the dependencies within a cloud. Over time, a cloud provider might lose potential customers due to the fuzziness calculated by the investigation, relent and sign a NDA with the CRR.

3 Our Approach

In this section, we delve into the inner workings of a CRR (cloud reliability recommender) as illustrated in Figure 3 or what effectively happens in Step 1.5 of Figure 2. With this information, we elaborate on how the CRR determines a recommended deployment for the consumer in the transaction shown in Steps 1 and 2 of Figure 2.

Step 1: Collecting dependency information. The CRR collects dependencies from various cloud providers as elaborated in Section 2.2. In short, this may be obtained by NDA, investigation, or by some other means.

Step 2: Generating the fault tree. After receiving dependency information, the CRR generates a fault tree for the set of service providers based upon service and hardware dependencies. As shown in Step 2 of Figure 3, the CRR builds fault tree for storage service A, B and C, respectively, and merges them together to build a composite tree. During the process of building the composite fault tree, our approach explicitly tags the common dependencies between redundancies to facilitate handling them in the next step.

Step 3: Discovering hidden common dependencies. The CRR finds all the common dependencies by traversing the fault tree generated in the previous step. Upon completion, the CRR generates a correlation matrix C, which reflects hidden correlations between different service providers. The elements in C, denoted as $\varepsilon_{i,j}$, rep-

resent the correlation coefficient between service i and j, computed by $\varepsilon_{i,j} = \delta/\lambda$ with δ representing the number of shared dependencies between i and j and λ denoting total number of elements that the i has.

Step 4: Computing failure probability for each service. The CRR can now calculate each service's failure probability through quantitative approach in fault tree analysis. Specifically, our approach begins by finding the *minimal cut sets* of the target fault tree. Minimal cut set here denotes the set constructed by the leaves which could directly make root fail. With the discovered minimal cut sets, our approach computes the failure probabilities of the target redundancies using top-event probability computation method [4, 17] which is designed based on conventional probability theory of occurrence.

Making recommendations to cloud consumers. The CRR now has sufficient information to handle consumer requests. While externally a CRR may only present a single deployment plan to the consumer, internally, it must rank the services in accordance with the cloud consumer's requirements. In order to recommend the most suitable deployment plan to a cloud consumer, a CRR makes use of Equation 1 to compute a *recommendation value* for each service in terms of the cloud consumer's requirements, where RV_i denotes service i's recommendation value. In our current design, we focus on the following three factors for calculating a rating score: failure probability (FP_i), correlation degree (CC_i) and cost (τ_i).

$$\begin{cases} RV_i = \alpha(1 - FP_i) + \beta(1 - CC_i) + \gamma(1 - \tau_i) \\ \alpha + \beta + \gamma = 1 \end{cases} \tag{1}$$

Where, FP_i denotes failure probability of provider i (computed in Step 4) and CC_i is the average of correlation coefficients for i and all other providers (obtained from correlation matrix generated in Step 3); τ_i denotes the cost of purchasing from i with a value within the range $[0, 1]$. α, β and γ are weighting coefficients specified by the cloud consumer for the deployment requirements. As mentioned in Section 2.1, this may either be explicitly stated or interpreted by the CRR. The weighting coefficients reflect the importance of three components (i.e., failure probability, correlation degree and cost) in computing final recommendation values. The range of Equation 1's result (i.e., recommendation value RV_i) is $[0, 1]$.

After computing all the services' recommendation values, the CRR produces a report consisting of the services ranked by their recommendation values. Depending on the CRR model, the CRR may send this report to the consumer, use this to deploy the consumer's services, or only recommend a specific deployment model using the "optimal" choices from the report and other criteria from the consumer. Sharing too much information may leak secrets breaking NDAs signed with providers or revealing the CRR's expensive independent research [11]. The process of transmitting this information to the consumer is shown in the second step of Figure 2.

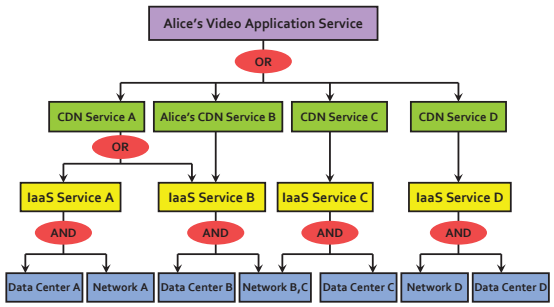

Figure 4: A case study illustrating our approach.

4 Case Study

In this section, we discuss a realistic, yet fictitious, case study similar to the Netflix situation mentioned in the introduction. In this scenario, Alice, a rising entrepreneur in video on demand (VoD), wants to deploy her services into the cloud using various content delivery networks (CDNs). Alice has many options at her disposal, as shown in Figure 4. During her investigations into options, she determines three potential providers, CDN Service A, C, and D, and also another possibility of building her own CDN using an IaaS, CDN Service B. Using Figure 4, she might decide to construct CDN Service B, herself, and subscribe to CDN services A and C for redundant services. Unbeknownst to her, that CDN Service A actually uses the same IaaS service used by her CDN, and that her CDN Service shares the same network as CDN Service C. A faulty Network B, C would invalidate the choice of CDN Service B and C, likewise, constructing CDN Service B only replicates software independence from CDN Service A but no infrastructure independence. Meanwhile she never considered CDN Service D, which happened to be completely resource independent from the other CDNs.

If Alice had been introduced to a cloud reliability recommender (CRR), she could have made a much more informed decision about her deployment. Now we illustrate a world in which Alice does know a CRR. Alice begins by telling the CRR her desire to use CDNs in a specific region along with the possibility of constructing her own CDN, CDN Service B in Figure 4. Taking this information, the CRR initiates the process of generating a deployment plan for Alice by building fault trees for the four potential providers, shown Figure 4. During analysis, the CRR discovers the existence of two common de-

pendencies: IaaS Service B and Network B, C. Using this information, the CRR calculates correlation coefficients: $\varepsilon_{A,B} = 1/2$, $\varepsilon_{A,C} = 1/4$, $\varepsilon_{B,A} = 1$, $\varepsilon_{B,C} = 1/2$, and $\varepsilon_{C,A} = \varepsilon_{C,B} = 1/2$, constructing the correlation matrix C, as shown in Equation 2, This mimics the process shown in the third step of Figure 3.

$$C = \begin{bmatrix} 1 & 1/2 & 1/4 & 0 \\ 1 & 1 & 1/2 & 0 \\ 1/2 & 1/2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Next, the CRR computes the failure probabilities for the four providers. For simplicity, we assume a data center's failure probability is 0.2, and a network service's failure probability is 0.1. Therefore, we get the failure probabilities for four CDNs as shown in Equation 3.

$$\begin{cases} FP_A = 1 - (0.72 + 0.72 - 0.72 * 0.72) = 0.0724 \\ FP_B = 1 - 0.9 * 0.8 = 0.28 \\ FP_C = 1 - 0.9 * 0.8 = 0.28 \\ FP_D = 1 - 0.9 * 0.8 = 0.28 \end{cases} \tag{3}$$

In order to recommend the most suitable deployment model to Alice, the CRR needs to compute the recommendation values for all the CDNs in terms of Equation 1. We assume that Alice cares more about reliability than correlated dependencies; thus, the CRR sets the weighting coefficient of reliability (FP) and correlation (CC) to 0.7 and 0.3, respectively, i.e., $\alpha = 0.7$ and $\beta = 0.3$. Moreover, Alice does not care about redundancies' cost and sets $\gamma = 0$. Therefore, the CRR calculates the recommendation values for the four CDNs:

$$\begin{cases} RV_A = 0.7 \cdot (1 - 0.0724) + 0.3 \cdot 0.75 = 0.87432 \\ RV_B = 0.7 \cdot (1 - 0.28) + 0.3 \cdot (1 - 0.5) = 0.654 \\ RV_C = 0.7 \cdot (1 - 0.28) + 0.3 \cdot (1 - 0.33) = 0.704 \\ RV_D = 0.7 \cdot (1 - 0.28) + 0.3 \cdot (1 - 0) = 0.804 \end{cases} \tag{4}$$

Using these computations, the CRR ranks the four providers based on their recommendation values. With additional criteria, such that Alice wants to use more than 1 but less than 4 CDNs, the CRR might suggest CDN A and D.

Interesting observations. There are a few interesting observations from this cast study: 1) if Alice cares significantly about shared dependencies (say, sets $\alpha = 0.2$ and $\beta = 0.8$), CDN D's recommendation value would be higher than A's, despite A's failure probability being the lowest; 2) even though the failure probabilities of CDN B, C and D are equal, the recommendation value for CDN D is much higher than the other two, due to the lack of common dependencies; and, 3) the current ranking ignores costs, which may further impact the rankings if taken into account.

5 Discussion

In this paper, we introduced the cloud reliability recommender (CRR), whose primary goal is to address the hidden risk in redundant cloud services. In arguing for the need for and use of a CRR, we discovered many challenges impeding its utility, but also many side effects that would benefit cloud consumers. The challenges internal to the CRR include the need to obtain private cloud information, a non-trivial task, and to calculate a recommendation for a consumer that both accurately takes into account their requirements (optionally) without revealing the CRR's private knowledge.

We have yet to address trust. Trust in this type of a system is paramount, not only between a client and CRR, but also a CRR and a cloud provider. Clients depend on the honesty of a CRR, if a CRR has been corrupted, it may provide deployment plans that are known faulty but generate revenue due to backroom agreements. Clients could request the service of many CRRs or the CRRs could reveal more information on how they established their model without compromising their insider knowledge. Likewise, a CRR could be played fool to a cloud provider who signs a NDA (non-disclosure agreement) but does not accurately describe their internal structure, such that they have multiple Internet connections without the fact that they all come from the same company. Again this could potentially be addressed by the CRR doing either independent investigation or seeing documentation that asserts the cloud provider's claims [19, 23].

In order to make our work more practical, we intend to do a deeper analysis on the use of fault trees and other fault tolerance analysis methods to represent data centers. Earlier researchers have looked at using fault tree analysis in computer systems [9, 13, 15]. This work may not be able to be trivially applied to a cloud data center, since such environments have significant overlaps in reliability, such as machine, storage, and networking redundancy, along with software techniques to recover from such failures.

Putting aside these issues, the CRR approach brings not only the benefit of more reliable deployments for consumers but also the opportunity to optimize along other guidelines. The emphasis of the paper is on reliability, but as we have glossed over throughout the paper, other aspects can impact a consumer's recommendation [**?**, 12, 18], such as cost and location. We envision a CRR may be generalized to a CR or cloud recommender.

References

[1] How Netflix, Zynga Beat Amazon Cloud Fail-

ure. http://www.informationweek.com/news/cloud-computing/infrastructure/232200511.

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

[3] Zynaga. https://zynga.com/.

[4] Tim Bedford and Roger Cooke. *Probabilistic Risk Analysis: Foundations and Methods*. Cambridge University Press, 2001.

[5] Alysson Neves Bessani, Miguel P. Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: dependable and secure storage in a cloud-of-clouds. In *EuroSys*, pages 31–46, 2011.

[6] Nicolas Bonvin, Thanasis G. Papaioannou, and Karl Aberer. A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In *SoCC*, 2010.

[7] Thierry Titcheu Chekam, Ennan Zhai, Zhenhua Li, Yong Cui, and Kui Ren. On the synchronization bottleneck of openstack swift-like cloud storage systems. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*, pages 1–9, 2016.

[8] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, pages 205–220, 2007.

[9] J.B. Dugan, S.J. Bavuso, and M.A. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *Reliability, IEEE Transactions on*, 41(3):363 –377, sep 1992.

[10] Bryan Ford. Icebergs in the clouds: the other risks of cloud computing. In *HotCloud*, 2012.

[11] Jianchun Jiang, Liping Ding, Ennan Zhai, and Ting Yu. Vrank: A context-aware approach to vulnerability scoring and ranking in SOA. In *Sixth International Conference on Software Security and Reliability, SERE 2012, Gaithersburg, Maryland, USA, 20-22 June 2012*, pages 61–70, 2012.

[12] Bo Liu, Ennan Zhai, Huiping Sun, Yelu Chen, and Zhong Chen. Filtering spam in social tagging system with dynamic behavior analysis. In *ASONAM*, July 2009.

[13] R. Manian, J. Bechta Dugan, D. Coppit, and K.J. Sullivan. Combining various solution techniques for dynamic fault tree analysis of computer systems. In *High-Assurance Systems Engineering Symposium, 1998. Proceedings. Third IEEE International*, pages 21 –28, nov 1998.

[14] Jeffrey C. Mogul. Emergent (mis)behavior vs. complex software systems. In *EuroSys*, pages 293–304, 2006.

[15] C. V. Ramamoorthy, G. S. Ho, and Y. W. Han. Fault tree analysis of computer systems. In *Proceedings of the June 13-16, 1977, national computer conference*, AFIPS '77, pages 13–17, New York, NY, USA, 1977. ACM.

[16] Mehul A. Shah, Mary Baker, Jeffrey C. Mogul, and Ram Swaminathan. Auditing to keep online storage services honest. In *HotOS*, 2007.

[17] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook*. U.S. Nuclear Regulatory Commission, January 1981.

[18] Yonggang Wang, Ennan Zhai, Jian-bin Hu, and Zhong Chen. Claper: Recommend classical papers to beginners. In *FSKD*, August 2010.

[19] Ennan Zhai, Ruichuan Chen, Zhuhua Cai, Long Zhang, Huiping Sun, Eng Keong Lua, Sihan Qing, Liyong Tang, and Zhong Chen. Sorcery: Could we make P2P content sharing systems robust to deceivers? In *9th P2P*, September 2009.

[20] Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. An untold story of redundant clouds: Making your service deployment truly reliable. In *9th HotDep*, November 2013.

[21] Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. Heading off correlated failures through Independence-as-a-service. In *11th OSDI*, October 2014.

[22] Ennan Zhai, Liang Gu, and Yumei Hai. A risk-evaluation assisted system for service selection. In *ICWS*, July 2015.

[23] Ennan Zhai, Huiping Sun, Sihan Qing, and Zhong Chen. Sorcery: Overcoming deceptive votes in P2P content sharing systems. *Peer-to-Peer Networking and Applications*, 4(2):178–191, 2011.

[24] Ennan Zhai, David Isaac Wolinsky, Hongda Xiao, Hongqiang Liu, Xueyuan Su, and Bryan 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.

[25] Ming Zhong, Kai Shen, and Joel I. Seiferas. Replication degree customization for high availability. In *EuroSys*, pages 55–68, 2008.