Resilience as a New System Engineering for Cloud Computing

Manghui Tu[1] and Dianxiang Xu[2]
[1]Department of CITG, Purdue University Calumet, Hammond, Indiana, 46323, USA
[2]College of Business and Information Systems, Dakota State University, Madison, South Dakota, 57042, USA
Email: manghui.tu@purduecal.edu; dianxiang.xu@dsu.edu

Abstract—It has become increasingly evident that large scale systems such as clouds can be brittle and may exhibit unpredictable behavior when faced with unexpected disturbances. Even weak and innocuous disturbances can bring down the system inoperative and may introduce catastrophic disasters to the society. The goal of this research is to explore the fundamental principles and theories that govern cloud system resilience and to provide novel and effective mechanisms to model and enhance the resilience of cloud. A food web like process interaction model is developed and system resilience enhancement mechanisms are proposed based on the control of the strength of interactions. Also, the effectiveness and limitations of modularization on resilience enhancement is illustrated by using a replica consistency control protocol. The research has shown that weakening key process interactions and modularizing complex systems are very effective on resilience enhancement.

Index Terms—system resilience; modeling; enhancement; cloud; interactions; modularization.

I. Introduction

Cloud computing has emerged as a new computing paradigm that delivers highly reliable and elastic services to satisfy users' dynamic demands in internet environment [1], [2]. The "*pay-as-you-use*" business model and computing elasticity have attracted huge attention from businesses and organizations [2]-[6]. With all the enthusiasm and excitement on the huge payoff cloud can bring into the computing world, there are still lots of hesitations and reluctances on adopting cloud computing due to the low confidence on the sustainability to components failures, security threats, operation and design mistakes, as well as nature disasters [7]-[11].

Cloud computing centralizes the management of many decentralized data centers across the world [4]. This vendor driven monoculture has achieved much higher degree of efficiency than traditional data centers and make it extremely attractive in cost efficiency. With its rigid operational parameters, resource redundancy, and elegant management system, cloud can achieve reduced failure rate as it is programmed to be. However, a fundamental problem is usually overlooked. In the cloud design and implementation, the system is designed to achieve high robustness under a small range of expected disturbances. Once an unexpected strong enough disturbance occurs, the cloud system cannot be resistant to such disturbance and this will first lead to local failures. Such failures may then be propagated to the rest of the system through the interdependent interactions in the hierarchical structure, causing a system wide failure. Despite numerous efforts spent on the research and practices on computer system and network security, system and data availability, business continuity, and fault tolerance, it has become increasingly evident that these large scale complex systems remain brittle and can exhibit unpredictable behavior when faced with unexpected disturbances, natural created or human made, intentionally or unintentionally [7], [9]-[12]. History has repeatedly shown that even weak and innocuous disturbances can shut the system down and may introduce catastrophic disasters to the society, which has been demonstrated by numerous case histories such as the 1940 Tacoma Narrow Bridge, 2005 Hurricane Katrina, the current economic crisis since Year 2008 [13]. The recent Amazon EC2 outage incident has brought such sustainability concerns back into the public's spotlight in computing world [14], [15]. A sustainable system are inherent resilient and thus a sustainable complex computing system should be designed with resilience, which is defined as the amount of disturbance that a system can survive without changing system state, including system's ability to absorb, be adaptive to, and to recover from the disturbance [5], [7], [11], [16]-[18]. System resilience has recently been studied for computing and network systems design [4], [5], [16]-[19]. Most of the current research works focus on analyzing the importance of resilience design [19], [20], resilience definition [16], resilient architecture design [16], [17], resilience assessment metrics [18], [19], [21], and the impacts of disturbance of information systems or local internet [18]. All of them focus on conceptual analysis or empirical measurements and some research works also provide suggestions on enhancing system resilience for networked systems such as P2P systems and the Internet [18], [21]. However, few of them have explored the fundamental principles and theories that govern complex computing systems' resilience property. Without a deep understanding on those fundamentals, it is unlikely to provide a systematic approach to effectively enhance the resilience of large scale complex systems such as clouds.

Manuscript received March 9, 2013; revised April 17, 2013.
Corresponding author email: manghui.tu@purduecal.edu.
doi:10.12720/jcm.8.4.267-274

The goal of this research is to explore the fundamental principles governing computer systems' resilience and to provide effective mechanisms enhance resilience for large scale systems such as cloud. The following objectives will be achieved. 1) The interactions among processes in the cloud will be modeled as a multi-level resource consuming hierarchy. Components of cloud will be modularized as autonomous sub systems and the effects of de-coupling of system components on system resilience will be analyzed. 2) A systematic approach will be developed to analyze and enhance system disturbance resilience for the cloud. 3) Tradeoffs among system resilience, robustness, efficiency, and performance will be investigated and effective resilience enhancement mechanisms will be developed.

The remainder of the paper is organized as follows. . A mini cloud system based on Eucalyptus is described in Section II. Process interaction modeling and resilience analysis based on interaction strength tuning are presented in Sections III. The effectiveness of modularization on system resilience enhancement is illustrated in Section IV. Section V concludes the paper.

II. System and Resilience Modeling

To understand the interactions among different components in the cloud, we implemented a mini cloud environment by using the Eucalyptus open source system [22], which can simulate the Amazon EC2's IaaS cloud environment (shown in Fig. 1).

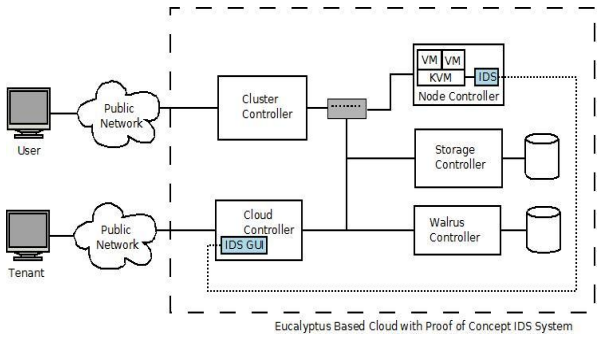

Figure 1. A mini eucalyptus cloud environment.

Eucalyptus provides a tiered design consisting of the cloud controller, storage controller, cluster controller, and node controller. The cloud controller provides the interface for tenants and administrators to query the node managers for resource information and makes high level scheduling decisions. The storage controller controls access to virtual machine images and tenant data. The cluster controller manages the virtual instance network and the scheduling of virtual machine execution on node controllers. The node controller is responsible for the execution, inspection and termination of virtual machine instances on the host where it runs [22]. The cloud controller, storage controller and cluster controller was installed on a single dual-NIC physical machine with an Intel Pentium 4 processor, two gigabytes of RAM, and a 500 gigabyte hard disk which resides on the public and private network. The node controller was installed on a system with an AMD Phenom quad core processor with AMD virtualization support, four gigabytes of RAM, and a terabyte hard disk.

System resilience is related to system robustness. Resilience emphasizes system conditions that are far from any stable steady-state, where disturbances can shift a system from one regime of behavior to another i.e., to another stability domain. Robustness is defined as the capacity of a system to maintain its performance when subjected to disturbances and can only maintain narrow band of states when exposed to disturbances [23]. Let H_o denote the system property of the original system state (denoted as S_o) and H_{min} denote the system property of the system state just before collapse (denoted as S_{min}). Hence, a cloud can be defined as *resilient* if H_{min} is far from H_o ($H_{min} << H_o$) and S_o is far from S_{min}, while a cloud can be defined as *robust* when $H_{min} \approx H_o$ and S_o is very close to S_{min}. The difference between system robustness and resilience can be illustrated in Fig. 2.

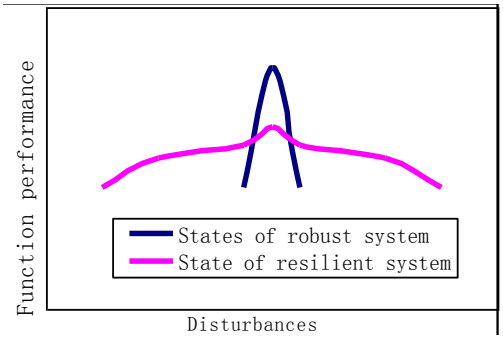

Figure 2. State of resilient system and robust system.

III. Resilience and Process Interactions in the Cloud

The recent Amazon EC2 outage [14], [15] is caused by the failure of a few components in the system. The analysis of such failure revealed that its weak resilience is causally related to the interactions among system components at different levels and the strength of interactions.

A. System Interaction Modeling

The architecture of cloud computing is service oriented and the interactions among cloud services can be modeled as a service-consumer model. In the cloud, a large number of business services are grouped into disjoint zones, and are then grouped into disjoint subregions, which are further grouped into disjoint regions. Each zone maintains a resource pool with a fixed capacity to serve business demands. The business services are managed by different sets of controlling services, which is organized into a tree network in order to adapt to the scale of the cloud. Each node in the tree is responsible for a single subregion, and it represents of a pool of controlling services to serve the requests from business services within such subregion. An instance of

controlling service may need to collaborate with other services, such as *utility services* at the same level, or to collaborate with the controlling services that are represented by neighboring nodes in the tree. Therefore, the services and their interactions in the cloud form a multi-level hierarchy, which can be shown in Fig. 3.

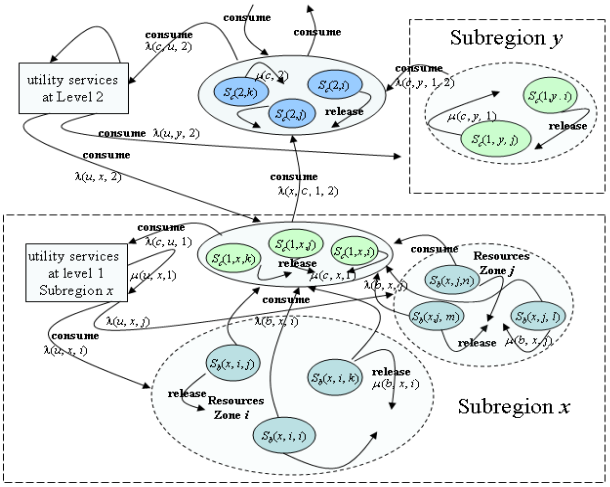

Figure 3. A preliminary model characterizing the interactions among processes in cloud

The interactions among services can be modeled in the same way as the food chains/food web predation modeling in natural ecosystems. Fig. 3 illustrates the service-consume relationship for a few components and processes in this system. In a cloud, each service will be accessed by a large number of consumers and the access requests are independent to each. Therefore, the arrival of the requests on a service hosted in the cloud can be modeled as a single Poisson Process with mean arrival rate λ [24]. For example, business services within zone i in Subregion x consume the control services that are responsible for Subregion x, at a rate of $\lambda(b, x, i)$; Controlling services in Subregion x consume other services such as the utility services in Subregion x, at a rate of $\lambda(c, u, 1)$; and consume controlling services at level 2 at a rate of $\lambda(x, c, 1, 2)$; Utility services in Subregion x consume resources in Zone i at a rate of $\lambda(u, x, i)$. Also, The expected service time for processing access requests on the service are about the same, thus, the service time of serving access requests can be modeled as exponentially distributed with expected service time of $1/\mu$. For example, resources are released in Zone i at a rate of $\mu(b, x, i)$, and controlling services in Subregion x are released at a rate of $\mu(c, x, 1)$. In the cloud system, a service may be requested by two or more types of consumer services. Since the requests from those consumer services to the consumed service are independent to each other, these arrival Poisson Processes of the requests can be combined into a single Poisson process with $\lambda = \sum \lambda_k$, where λ_k is the arrival rate of requests from each consuming service k. Also, the expected service time for processing requests from different types of services are about the same, the service time can be modeled to be exponentially distributed with expected service time of $1/\mu$. Based on queuing theory, the interactions between service and consumers can be modeled as an $M/M/1$ Markov process, with arrival rate of λ and service rates of μ.

Under normal conditions, these $M/M/1$ Markov processes will be in a dynamic equilibrium state such that $\lambda(b, x, i) + \lambda(u, x, 2) < \mu(c, x, 1)$, $\lambda(x, c, 1, 2) + \lambda(y, c, 1, 2) < \mu(c,2)$. However, if there is an unexpected disturbance affecting a business service $S_b(x, i, k)$ in zone i, $\lambda(b, x, i)$ can reach a very high level such that $\lambda(b, x, i) > \mu(c, x, 1)$. According to queuing theory, the controlling services will soon be exhausted and this will lead to the unavailability of service in zone i and then the unavailability of all business services in Subregion x. This cascading phenomenon can be much worse if it involves with controlling services at upper levels, which could shut down the entire cloud

B. System Resilience Analysis

The resilience of a system is a property very difficult to measure since it could be affected by many factors. The strength of resilience can be demonstrated only when there is a strong enough unexpected disturbance. Also, under different conditions, the impact of resilience can be very different. In this research, system resilience is defined as the number of requests to business services (denoted as π) in the system that could be affected and even failed, by a given disturbance Ψ. Also, a system with smaller number of affected requests to business services is considered to have a stronger resilience. The strength of a disturbance Ψ, denoted K^{Ψ}, can be measured as the change of the rate (denoted as $\Delta\lambda$) that a services consumes the another services and the time period t such change lasts, i.e., $K^{\Psi}(\eta) = f_{\Psi}(\eta, \Delta\lambda, t)$. Let $\eta^{A,B}$ denote the interaction between service A and service B (A consumes B) and let $P^{A,B} \in [0, 1.0]$ denote the strength of $\eta^{A,B}$, which is measured as the maximum proportion of B that can be consumed by A, and the value of P is independent of the consuming rate and disturbance strength K^{Ψ}. Note that the actual proportion of B that will be consumed by A depends on the consuming rate λ. Based on the definition, the value of π can be measured as function of interaction strength and disturbance strength. For simplicity and demonstration purpose, let's only consider a single interaction $\eta^{A,B}$ in the system, then system resilience can be defined as Eq.1,

$$\pi = f_{\pi}(K^{\Psi}(\eta^{A,B}), P^{A,B}) = f_{\pi}(f_{\Psi}(\eta^{A,B}, \Delta\lambda, t), P^{A,B}) \quad (1)$$

It can be inferred from Eq. 1 that system resilience is causally related with the strength of system interaction, and we define their causal relationship in Theorem I.

Theorem I: Given a system S with an arbitrary interaction $\eta^{A,B}$ and an arbitrary strong disturbance Ψ with a strength of $K^{\Psi}(\eta^{A,B})$. Let S_1 denote the system S designed with interaction $\eta^{A,B}{}_1$ and S_2 denote the system S

designed with interaction $\eta^{A,B}{}_2$, where S_1 differs from S_2 only on the strength of interaction $\eta^{A,B}$. If $P^{A,B}{}_1 < P^{A,B}{}_2$, then $\pi_1 < \pi_2$, when S is facing Ψ with the strength of $K^{\Psi}(\eta)$.

Proof: Let the cloud system be S, business services in Zone i be A and let the controlling service in the Subregion x be B, then the service-consuming relationship between is $\eta^{A,B}$. Let C_x denote the total number of controlling service instances in Subregion x, N_i be the number of business service instances in zone i, N_x denote the total number of business service instances that are served by the controlling services in Subregion x, we have $N_i < N_x$. The two designs for S (S_1 and S_2 differs only on the relationship between the controlling service Subregion x and business service in Zone i. In design S_1, only C_i instances of controlling services are designated to serve business services in Zone i, where $C_i = C_x * N_i / N_x$, then we have $C_i < C_x$ and $P^{A,B}{}_1 = C_i / C_x < 1$. An $M/M/K$ (a general form of $M/M/1$) queuing system can be used to analyze the state dynamics of the interaction $\eta^{A,B}{}_1$ in design S_1. A high $\lambda(b, x, i)$ introduced by Ψ will result in a huge number of requests in the queue since the service rate of the controlling service, $\mu(b, x, i)$, is constant. This will exhaust all C_i instances of controlling services which cannot provide to services to new requests. Hence, the affected number of business service instances is N_i, i.e., $\pi_1 = N_i$. In design S_2, all of C_x instances of controlling services can be allocated to serve business services in Zone i, then we have $P^{A,B}{}_2 = C_x / C_x \equiv 1$. Similarly, a high $\lambda(b, x, i)$ introduced by Ψ can result in a huge number of requests in the queue and it will eventually exhaust all C_x instances of controlling services. Thus, the controlling services in Subregion x cannot provide services to any new business requests in the entire Subregion x. Therefore, the affected number of business service instances is N_x, i.e., $\pi_2 = N_x$. Since $N_i < N_x$, it follows that $\pi_1 < \pi_2$, also, $P^{A,B}{}_1 < 1 \equiv P^{A,B}{}_2$. Thus, Theorem I follows.

However, there are a few challenges need to be addressed before we can design mechanism to tuning the strength of interactions. Specifically, the impacts of the existence of other interactions on an interaction $\eta^{A,B}$ and system resilience should be investigated and modeled in the following four cases.

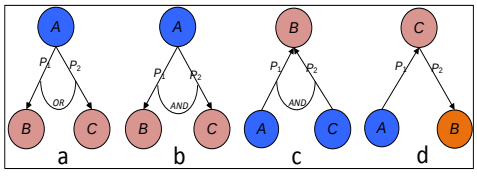

Figure 4. Relationships of consumers and resources.

Case I: The existence of an alternative interaction $\eta^{A,C}$, shown in Fig. 4 (a). C provides resource redundancy to A and will help Service A to maintain its persistent functions, when the availability of resource B becomes low. Thus, to improve system resilience, it is desired to provide service redundancy.

Case II: The existence of a concurrent interaction $\eta^{A,C}$, shown in Fig. 4 (b). The concurrent resource C required by Service A will hurt the persistent behavior of Service A, when faced with disturbances and the availability of resource C becomes low. Therefore, to improve system resilience, it is desired to break or avoid such interdependency between the interaction $\eta^{A,B}$ and the interaction $\eta^{A,C}$.

Case III: The existence of a competitive interaction $\eta^{C,B}$, shown in Fig. 4 (c). The competitive consumer C will definitely hurt Service A to provide persistent service to users, when faced with disturbances that the arrival rate of the requests from C increased. Thus, to improve system resilience, it is desired to break or mitigate the strength of the competitive relationships among interactions.

Case IV: The existence of chained interactions $\eta^{A,C}$ and $\eta^{C,B}$, shown in Fig. 4 (d). The interactions $\eta^{A,C}$ and $\eta^{C,B}$ will increase the degree of dependence of components on each other. When faced with disturbance, the low availability of B or C will hurt the persistence of Service A. Furthermore, if $\eta^{A,C}$ and $\eta^{C,B}$ are at different levels of the multi-level hierarchy, then a disturbance leading to high request rate of A may introduce disruptions into higher levels of the system, and thus affect more services in the system. Hence, to improve system resilience, it is desired to broke such chained interactions of $\eta^{A,B}$ and $\eta^{A,C}$.

C. System Resilience Enhancement

In a natural ecosystem, species that sustain or flourish in an unstable environment are r-selected, i.e., focusing more on growth rate by exploiting redundant resource to generate more offspring [25]. To improve system resilience, a straightforward approach is to provide extremely high redundancy of the services or resources to be consumed. For example, a cloud can grant highly redundant controlling services for each zone and subregion, which can help to maintain normal functions even under strong disturbances. However, in a natural ecosystem, r-selected species usually cannot sustain or flourish in highly competitive stable environment, and will eventually be replaced by K-selected species (focusing more on performance and efficiency) [25]. If a cloud is designed with too much redundancy, then it will sacrifice too much cost efficiency. Obviously, such a cloud will not be able compete with other cloud vendors and will eventually be driven out of the market. Therefore, an ideal resilient cloud should be designed with the balance of the two strategies.

With certain resource redundancy, system resilience can also be enhanced by weakening the strength of interactions (i.e., lowering the value of P). The value of P can be lowered in such a way that each pool of consumers can have abundant resources reserved to maintain acceptable performance, i.e, to achieve the acceptable response time specified in user's *SLA* (D_s). To achieve

high efficiency and performance under stable environment, a shared pool of resources is maintained to achieve good response time ($D_e \geqslant D_s$). In this way, the cloud can achieve good resilience, performance, and efficiency at the same time. To determine the number of reserved controlling services (k) for each type of cloud applications (business services), the interactions among them are approximately modeled as an $M/M/K$ queuing system. Hence, k can be computed by using Erlang C Formula, e.g.,

$$D_e = (1+\alpha)/(\mu*(k-\lambda/\mu))$$

where α is the probability that all k services are busy and α is determined by k, λ and μ. Theoretically, with this mechanism, the impact of unexpected disturbance can be contained without the loss of efficiency on the consuming of controlling resources.

IV. System Resilience and Modularization

It is well known that modularity plays a critical role in system robustness and resilience at different levels in biological systems [9], [20], [26]. A modularized system can provide strong resilience since local failures can be isolated and contained [9], [20]. In such a modularized system, modules are integrated and communicated through management protocols, which define the corresponding architectures, rules, interfaces, etiquettes, and codes of conduct for modules [9], [26]. When subjected to disturbances, bad-designed protocols can be vulnerable [9], [26]. Macroeconomists and scientist in system engineering have attempted to apply modularity to improve system resilience [9].

In this research, the impact of modularity on cloud resilience is illustrated by the epidemic based replica consistency control protocol [27]. Data objects can be replicated across the cloud to serve user access and the consistency of replicas can be controlled by using an epidemic based update protocol [27]. However, high number of conflicting updates may lead the cloud into a non-recoverable state, which will lead to the crash of the entire system [27]. To improve system's robustness to such disturbance, the components modularity mechanism has been applied.

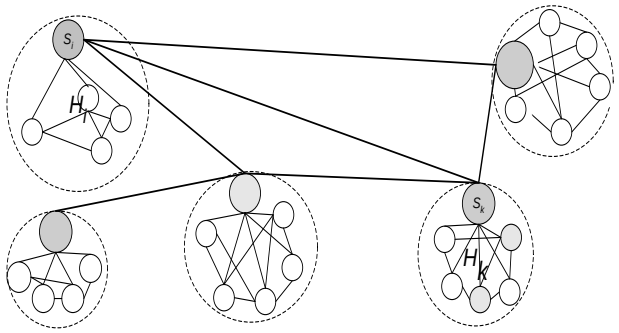

Figure 5. Topology of the replicas in the cloud.

Replica sites are clustered into non-overlap local groups. Each local group will have a super site that represents the local group to communicate with other groups for update exchange. The set of super sites are organized into a super group. The replica sites can only disseminate updates directly with each other within the same group (either in a local or a super group). The modularized system results in a two level hierarch, which is shown in Fig. 5.

Let site(t) denote the peer at which the transaction t is first executed, $TS(t)$ denote the timestamps of t, $RS(t)$ and $WS(t)$ denote the read and update data set. When a user needs to access a data object d, it can access d at any peer that holds a replica of d. For an update t, site(t) needs to propagate it to all other peers that hold a copy of any non-empty subset of $WS(t)$. Each replica site P_i keeps a timetable T_i (shown in Fig. 6), each row of which, e.g., $T_i[k, *]$, represents P_i's knowledge of the updates received at peer P_k. If $T_i[k, j] = v$, then P_i knows that P_k has received the v^{th} update (namely, t') that is originally executed at S_j and all other updates that are causally preceding t'. The row $T_i[i, *]$ represents P_i's record of the received updates that are originally issued at each replica site, e.g., $T_i[i, j] = u$ means that P_i has received the u^{th} update (namely, t) that is issued at P_j and all other updates that are causally preceding t.

In the timetable, the k^{th} row of T_i is P_i's knowledge about P_k's reception of updates in the system, i.e., $HasRecvd(T_i, t, P_k) \equiv (T_i[k, \text{site}(t)] \geq TS(t)[\text{site}(t)])$. When P_i executes an update, it places a record in the log. When P_i sends a message to P_k, it includes all of such update t that $HasRecvd(T_i, t, P_k)$ is false, together with T_i. When P_i receives a message from P_k it applies all non-conflicting updates and updates its time-table in an atomic step to reflect the new information received from P_k. When a site receives a log record, it knows that the log records of all causally preceding events either were received in previous messages, or are included in the current message. When P_j receives an update t issued at P_i, it first searches its local log L_j to see if there exists such a transaction t' that $TS(t) <> TS(t')$ (t and t' are executed concurrently), and the data accessed are overlapping (i.e., $(WS(t) \cap WS(t') \neq \phi) \vee (RS(t) \cap WS(t') \neq \phi) \vee (WS(t) \cap \text{RS}(t') \neq \phi)$ (if only consider transactions accessing a single data object and allow users to read old data, then the condition of $(WS(t) \cap WS(t') \neq \phi)$ is sufficient). If such a transaction t' exists, then a conflicting flag is set with $r(t)$, and both t and t' will be aborted.

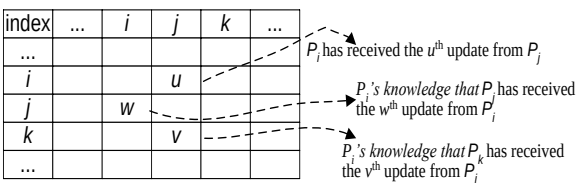

Figure 6. Sample timetable T_i at peer P_i.

```
1 ReceiveLocal (P_{k,i}, msg, P_{k,j}){
2 updateSet = msg. updateSet;
3 T_{k,j} = msg. timetable;
4 for ∀ {t| r(t)∈ updateSet ∧ ¬HasRecvd(T_{k,i}, t, P_{k,i})}{
5 begin mutex
6 if {∃ t'∈ L_{k,i}|(TS(t)<>TS(t'))∧(WS(t) ∧ WS(t') ≠ ϕ) }{
7       resolveConflict(t);
8       t.inconflict = true;
9       if (site(t') ∈ H_k) { resolveConflict(t');
10      t'.inconflict = true;}
11      if (site(t') belongs to H_k) t'.inconflict = true;      }
12 if {∃ t'∈ L_{k,i}| (t'.inconflict) ∧(t read from t')
              ∧(WS(t) ∧ WS(t') ≠ ϕ) } {
13      resolveConflict(t);
14      t.inconflict = true;                }
15 else if ((¬ t.inconflict) && (P_{k,i} holds WS(t)) {
16      acquire write lock on WS(t)
17 execute update t locally;
18 commit(t) and release write lock;
19 If (i≡0) { // super peer, it must hold WS(t)
20    T^S_k[k, k] = T^S_k[k, k] + 1;
21    t^S = t;
22    TS(t^S) = T^S_k[k, *];
23    site(t^S) = S_k;
24    create r(t^S) by appending TS(t^S) ;
25    L^S_k = L^S_k ∪ {r(t^S)};}// end super peer}
26  if (T_{k,i}[i, site(t)] < TS(t)[site(t)])
          T_{k,i}[i, site(t)] = TS(t)[site(t)] ;
27    L_{k,i}= L_{k,i} ∪ {r(t)};
28  end mutex }; // end foreach
29  begin mutex
30    ∀ m,n, T_{k,i}[m, n]= max (T_{k,i}[m, n], T_{k,j}[m, n]);
31     L_{k,i} = {t | r(t)∈ L_{k,i} ∧ ∃ n| HasRecvd(T_{k,i}, t, P_{k,n})};
32  end mutex}
```

Figure 7. The protocol for a site $P_{k,i}$ handling received updates from another site $P_{k,j}$ from the same local group.

The consistency protocol at a local group site with modularization is shown in Fig. 7 and the section of codes deals with modularization is in bold font. Once a super site $P_{k,i}$ ($i \equiv 0$) receives an update from a local site $P_{k,j}$ (at line 20), it will first change the timestamps of t for the super group (line 20-22), update the new update generation site for t (line 23), creates a new update record of t for the super group (line 24) and incorporates the new update record into the log used for the super group (line 25).

A. *Effectiveness of Modularization on System Resilience Enhancement*

To evaluate the effectiveness of modularization on the improvement of system resilience (locally executed updates that are conflicting with updates executed at other sites), the two level update protocol used for the modularized system is compared with classic one level update dissemination protocol [27].

The number of *conflicting units* (the sum of the conflicting transactions that all sites have received, whenever a conflicting is detected) is used as the parameter to evaluate the impact of the disturbance on the system. When there is a higher number of conflicting transactions introduced in the system, the higher the possibility of system will reach the state that cannot be recovered from the inconsistency and lead the crash of the entire system. Results are shown in Fig. 8 (a) and Fig. 8(b).

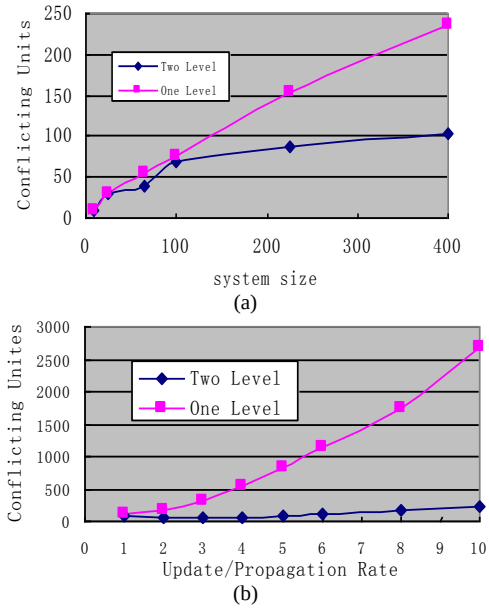

Figure 8. (a) Impact of system size and (b) Impact of update propagation rate.

From Fig. 8 (a) and Fig. 8 (b), it can be seen that as the system size and the update propagation rate increase, the impact of modularization on system resilience become more significant. This is because that with modularization, each local group consists of less number of replica sites, thus conflicting is much easier and earlier to be detected within the local group. Also, the conflicting updates originally executed at different local groups will be forwarded to the super group and the conflicting will be detected by the super group before they are forwarded to other local groups. All of these will create "*isolated islands*" for conflicting update. The larger the update propagation rate, the larger the number of replica sites will receive the conflicting updates, and the two-level update protocol can detect such conflicting much earlier than the one-level update protocol. The smaller the update propagation rate, the smaller the number of replica sites will receive those conflicting updates, hence, the effectiveness of the two level update protocol on preventing inconsistent update propagation is close to that of the one level update protocol. It can make a preliminary conclusion that modularization is very effective on the improvement of system resilience to unintentional disturbances.

B. *Limitation of Modularization on System Resilience Enhancement*

Inappropriate modularization may create new barriers for communications among the modularized sub systems. A single path between two sub systems in the cloud may make the system vulnerable to single point of failure and denial of service attacks. Thus, appropriate redundancy on components and pathways is critically needed. Also, the topology of the network composed by the

modularized system may have a great impact on system's resilience. Disassortative networks, in which a few super nodes connect many regular nodes, while regular nodes do not connect many other nodes, have shown strong resilience to disturbances. Therefore, the relationships among modularization, resource redundancy, and resilience need further study.

With modularization, distributed protocols are required to enable sub systems to collaborate. Elegant protocols designed with efficiency and accuracy in mind may be vulnerable to sophisticated intentional attacks [18]. In the modularized system described in [27], a single compromised replica site can manipulate the timestamps of an update that is executed locally. For example, a compromised peer P_3 can creates a timestamps $TS'(t)$ = (1, 3, **1**, 5, 4) for the update t whose original $TS(t)$ is (1, 3, **3**, 5, 4), as shown in Fig. 9. Without detection, such attack may introduce non-existent conflicts into the system. Examples of such updates could be t_2 and t_3 with $TS(t_2)$ = (1, 3, **3**, 4, 4) and $TS(t_3)$ = (1, 3, **2**, 4, 4), and t_2 (or t_3) and t update on the same data object. With $TS(t)$ manipulated, t_2 (or t_3) and t will conflict with each other, but actually their relationship should be that t_2 (or t_3) precedes t.

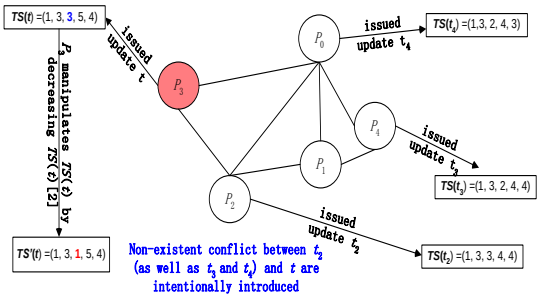

Figure 9. $TS(t)$ is manipulated by the compromised peer P_3 by decreasing $TS(t)$[2].

This type of manipulation can introduce a large number of conflicting updates in the system, which may lead to the crash of the entire system [27]. Such a denial of service attack exploits the vulnerabilities existed in the update propagation protocol. It successes only if it can successfully exploit multiple vulnerabilities in the system at the same time. A well-known model for multiple sources of disturbance is the Swiss Cheese Model [16]. In such system, each layer or component is resilient to the intentional attacks but may exhibit one hole. An intentional disturbance (attack) penetrates the hole on each layer can eventually affect the entire system, which leads to the loss of system resilience.

Therefore, to design the system resilient to such disturbance, monitoring mechanisms with detection and adaptive features should be designed. Thus, any penetration of such hole can be monitored and detected, and systems can be adapted to remove or disassociate such compromised layers or components. In research described in [27], it has been shown that a well-designed management protocol with monitoring and detection features can achieve strong resilience to such intentional disturbance. However, the gain in resilience comes with the loss of function performance and may require redundant resources. Hence, modeling the tradeoffs among system resilience gain, performance loss, and resources redundancy is critical for large scale systems such as cloud.

V. Conclusions

A In this paper, we explored the fundamental principles and theories that govern cloud system resilience and provided novel and effective mechanisms to model and enhance the resilience of cloud. To study the interactions among the processes in the cloud, a mini cloud system is implemented by using the Eucalyptus Open Source Software. A food web like process interaction model is developed and the interactions are modeled as *M*/*M*/1 queuing systems. The strength of interactions is defined and system resilience enhancement mechanisms are proposed based on the control of the strength of interactions. Also, the effectiveness and limitations of modularization on resilience enhancement is illustrated by using a replica consistency control protocol and experimental studies are conducted to measure the effectiveness of modularization. The research has shown that weakening key process interactions and modularizing complex systems are very effective on resilience enhancement.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, *et al.*, "Above the Clouds: A Berkeley view of Cloud Computing," University of California, Berkeley, Tech. Rep., 2009.

[2] P. Melland and T. Grance. (2009). *The NIST Definition of Cloud Computing*. [Online]. Available: http://csrc.nist.gov/groups/SNS/cloud-computing/

[3] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-senstive application performance in the cloud," in *Proc.* 1st *ACM Multimedia Systems Conference*, 2010.

[4] G. Briscoe and A. Marinos, "Digital ecosystems in the clouds: towards community cloud computing," presented at the 3rd IEEE Digital Ecosystems and Technologies Conference, 2009.

[5] G. Brunette and R. Mogull. (2009). Security Guidance for Critical Areas of Focus in Cloud Computing V2. 1. CSA (Cloud Security Alliance), USA. [Online]. Available: http://www.cloudsecurityalliance.org/guidance/csaguide

[6] D. Owens, "Securing elasticity in the cloud," *Communications of the ACM*, vol. 53, no. 6, 2010.

[7] D. Aresenault and A. Sood, "Resilience: A systems design imperative," in *Critical Infrastructure Protection Program Discussion Paper Series*, George Mason University, 2007.

[8] D. Gribble, "Robustness in complex systems," in *Proc. Eighth Workshop Hot Topics in Operating Systems*, May 2001, pp. 21-26.

[9] J. Kambhu, S. Weidman, and N. Krishnan, "(Rapporteurs) New directions for understanding systemic risk," *Econ. Policy Rev*, vol. 13, no. 2. 2007.

[10] E. Messmer. Are Security Issues Delaying Adoption of Cloud Computing? [Online]. Available: http://www.networkworld.com/news/2009/042709-burning-security-cloud-computing.html

[11] L. Perelman, "Shifting security paradigms: toward resilience," in *Critical Infrastructure Protection Program Discussion Paper Series*, George Mason University, 2007.
[12] H. Kitano, "Towards a theory of biological robustness," *Journal of Molecular Systems Biology*, vol. 3, no. 137. 2007.
[13] F. Schweitzer, G. Fagiolo, D. Sornette, F. Vega-Redondo, A. Vespignani, and D. White, "Economic networks: The new challenges," *Science*, vol. 325, 2009.
[14] Amazon, "Amazon Web services overview of security processes," *Amazon White Paper*, May 2011.
[15] Amazon. (May 2011). Summary of the Amazon EC2 and Amazon RDS service Disruption in the US East Region. [Online]. Available: http://aws.amazon.com/message/65648/
[16] S. Jackson, *Architecting Resilient Systems*, ISBN 978-0-470-40503-1. Wiley. 2010.
[17] D. Liu, R. Deters, and W. Zhang, "Architecture design for resiliency," *Journal of Enterprise Information Systems*, 1-16, iFirst article. 2009.
[18] M. Omer, R. Nilchiani, and A. Mostashari, "Measuring the resilience of the global internet infrastructure system," in *Proc. 3rd Annual IEEE International Systems Conference*, 2009.
[19] D. Garbin and J. Shortle, "Critical thinking: moving from infrastructure protection to infrastructure resiliency," in *Critical Infrastructure Protection Program Discussion Paper Series*. George Mason University, 2007.
[20] R. May, S. Levin, and G. Sugihara, "Complex systems: Ecology for bankers," *Nature*, vol. 451, pp. 893, February 2008.
[21] S. Wang, D. Xuan, and W. Zhao, "Analyzing and enhancing the resilience of structured peer-to-peer systems," *Journal of Parallel and Distributed Computing*, vol. 65, no. 2, 2005.
[22] D. Nurmi, *et al*., "The eucalyptus open-source cloud-computing system," *IEEE Computer Society*, 2009.
[23] D. Read, "Some Observations on Resilience and Robustness in Human Systems," *Cybernetics and Systems*, vol. 36, 2005.
[24] M. Tu, H. Ma, I. Yen, and D. Xu, "Data placement in P2P data grids considering the availability, security, access performance and load balancing," *Journal of Grid Computing*, vol. 11, no. 1, 2013.
[25] M. Janssen and J. Anderies, "Robustness tradeoffs in social ecological systems," *International Journal of the Commons*, vol. 1, no. 1, 2007.
[26] M. Csete and J. Doyle, "Reverse engineering of biological complexity," *Science*, vol. 295, March 2002.
[27] M. Tu, Z. Xia, and D. Xu, "Securing epidemic based update protocol in P2P data grids," In *Proc. PDCS11*, December 2011.

Manghui Tu received the PhD degree in computer science from the University of Texas at Dallas in 2006. He is currently an assistant professor in Department of Computer Information Technology and Graphics at Purdue University Calumet. He was an assistant professor of computer science at Southern Utah University from 2006 to 2009 and assistant professor of information assurance at Dakota State University from 2009-2012. His research interests include sustainable computing, distributed systems, information assurance, and digital forensics. He is a member of IEEE.

Dianxiang Xu received the BS, MS, and PhD degrees in computer science from Nanjing University, China. He is an associate professor with the National Center for the Protection of the Financial Infrastructure, Dakota State University, South Dakota. He was an assistant professor of computer science at North Dakota State University from July 2003 to May 2009, research assistant professor and engineer of computer science at Texas A&M University from August 2000 to July 2003, and research associate at Florida International University from May 1999 to August 2000. His research interests include software security and safety, software testing, applied formal methods, and computer forensics. He is a senior member of the IEEE.