A Petri Net Model for Secure and Fault-Tolerant Cloud-Based Information Storage

Daniel F. Fitch and Haiping Xu
Computer and Information Science Department
University of Massachusetts Dartmouth, North Dartmouth, MA 02747, USA
{daniel.fitch, hxu}@umassd.edu

***Abstract*—Cloud computing provides a promising opportunity for both small and large organizations to transition from traditional data centers to cloud services, where the organizations can be more concerned with their applications, services, and data rather than the underlying network infrastructures and their associated cost. There are major concerns, however, with data security, reliability, and availability in the cloud. In this paper, we address these concerns by proposing a novel security mechanism for secure and fault-tolerant cloud-based information storage. We present a formal model of the security mechanism using colored Petri nets (CPN). The model utilizes multiple cloud service providers as a cloud cluster for information storage, and a service directory for management of the cloud clusters including service query, key management, and cluster restoration. Our approach not only supports maintaining the confidentiality of the stored data, but also ensures that the failure or compromise of an individual cloud provider in a cloud cluster will not result in a compromise of the overall data set.**

Keywords-Cloud computing; information storage; data security; fault tolerant; colored Petri nets; formal modeling and verification.

I. INTRODUCTION

As the Internet continues to evolve, service-oriented systems are becoming more widely adopted by large companies and government into their computing platforms. Cloud computing extends this concept, allowing for access to powerful, ubiquitous and reliable computing to the general public through the use of web services and application programming interfaces (API). Although there is a large push towards cloud computing, there is a lack of work having been done in regards to data security, ownership and privacy in cloud computing. A survey conducted by the US Government Accountability Office (GAO) states that "22 of 24 major federal agencies reported that they were either concerned or very concerned about the potential information security risks associated with cloud computing" [1]. Due to the infancy of cloud computing, there are not many standards or best practices in terms of securing data in the clouds. Besides a few major companies investing into the cloud, there are also numerous startups and smaller companies attempting to become cloud providers. For these smaller entities, there are no guarantees that they are following or have the resources available to follow best practices for securing their data centers. In addition, services change frequently as product offerings are developed and discontinued, leaving users of the service scrambling to find alternatives, forced to take a migration path etched by the provider, or stuck using a service that is no longer being developed, refined, and patched. In an enterprise environment, the issues that are plaguing cloud computing would be considered unacceptable. If the corporate world is to adopt cloud computing, it must guarantee that the stored data is secure, stable, and available in the cloud.

Personal Information (PI), such as credit card information, that falls under Payment Card Industry (PCI) data security standards legislation, or medical records that falls under the Health Insurance Portability and Accountability Act (HIPAA), are especially at question regarding if and how exactly these pieces of data can be stored and managed utilizing cloud computing. Although there are some attempts to address HIPAA compliance in the cloud [2], there exist no widely accepted best practices or clear recommendations as to how this data can be stored in the cloud. Currently, users are advised to seek their own legal counsel on this matter, with the provider offering no liability for misguiding or incorrect advice. In addition, legislative acts, such as HIPAA, were developed with traditional network architectures in mind, with numerous regulations regarding physical facilities, employee best practices, and operating system best practices. In a cloud environment, all or at least most of these implementation details are hidden from the cloud consumers, so companies that fall under HIPAA regulations do not have direct influence or authority over these compliance details. Procedures and requests can be specified during the contract creation time between a cloud provider and an enterprise, but this would require complex negations and audit procedures that the cloud provider may not be equipped for or willing to follow. There are efforts in the security industry to certify certain cloud providers as being compliant with legislative mandates for handling PI, but we can find no established practice at this time. In this paper, we develop a security model that addresses these issues of cloud computing, easing concerns of legislatures and enterprise of storing data in the cloud. The proposed model can be adopted to serve as an equivalent alternative to enterprise controlled facility, personnel and infrastructure mandates.

Although cloud computing is still in its infancy, there has been a considerable amount of work on data security and federation for distributed data, in which this work is related to. Goodrich *et al.* explored efficient authentication mechanisms for web services with unknown providers [3]. In their approach, they utilized a third party notary service that could

ensure users the trustworthiness of the service providers. Weaver studied the topic of exploring data security when using web services [4]. In his approach, he placed a layer of authentication and authorization services between the clients and the web services they were trying to access in order to authorize users. He also explored the issue of federation to manage trust relationships among services, which could be extended towards securing cloud computing. Santos *et al.* provided efforts to establish a secure and trusted cloud computing environment [5]. They, however, assumed that providers could prevent physical attacks to their servers, which might not be true in the case of a poorly vetted employee or a poorly designed facility. In our approach, we focus on data security, redundancy, and privacy issues in cloud computing. We develop a formal model of information storage that utilizes a cloud cluster with multiple cloud providers to leverage the benefits of storing data in the cloud while minimizing the risks associated with storing data in an offsite, insecure, unregulated and possibly noncompliant atmosphere.

II. Secure and Fault-Tolerant Cloud-Based Information Storage

A. A Motivating Example

Consider a scenario where a medical company wishes to have all medical records of its patients available to its trusted partners, as well as to doctors who may be off site. First, the medical data is required to be highly fault tolerant, as losing patient records is not an option. Secondly, the data must be secure, as the company has an obligation to its patients to protect their personal information. Thirdly, the medical records must be guaranteed to be available, as it may become a matter of life or death if the data cannot be accessed quickly. The company realizes that storing the data on site would require a complex setup to make the data widely available to its central location and branch offices, with a large cost to purchase servers and storage devices. Furthermore, storing the data on site also requires a robust mechanism to ensure that the data is redundant and available in case of disaster, as well as a scalable infrastructure in case of growth. The company is attracted by the benefits of cloud computing, namely the availability of the data over the Internet for its remote offices and doctors, not having to invest a large amount of money to establish the infrastructure, the scalability, and the promise of resiliency and redundancy. Therefore, the company wishes to explore the option of using cloud computing for information storage and archiving of its data. It is, however, very concerned with moving its data into the cloud since losing physical control of its data be of high risk. Although the company can choose reputable cloud providers to host its data, there is no way to vet individual employees who are hired by the cloud provider to prevent insider attacks, whereas the medical company is required to do full background checks and audits on employees who are allowed to handle its data. The company is also concerned with the physical locations of its data. With a cloud provider, the medical company does not even know where its data resides in the cloud, let alone what safeguards are at place at the physical facility. The company has also seen through the media the amount of damage that can be caused by its data being compromised by a third party. It must assume that by storing its data in the cloud, it can be compromised, so it needs to ensure that the cloud providers and their employees absolutely do not have access to the underlying information. The company is finally concerned with the availability of its data, as although it sees cloud computing as mostly reliable, it needs to make sure that its data is available and that there are no extended length of outages. When the medical company is treating a patient, for example, it is critical to know if the patient is allergic to any medication. If this information became unavailable, there may be dire consequences. If the company chooses to build the data center by itself, it would take into account all of the above concerns. In the cloud, however, the environment is ever-changing with providers having the ability to decide critical implementation details, where data resides, and can at whim discontinue or radically change a service offering. Given the current state of cloud computing, the healthcare provider would have some very serious and legitimate concerns that need to be addressed.

In order to mitigate the major concerns that the medical company faces, we design a reliable, fault-tolerant, and secure architecture for cloud computing. Our approach can assist in bridging a major issue that resides in cloud computing yet to be solved, namely how to securely store personal information in the clouds. Thus, our approach can mitigate concerns from companies that are trying to adopt cloud computing and regulators as well as the general public who are concerned for the security and confidentiality of the stored information.

B. An Architectural Design

Our proposed information storage model for cloud computing can be deployed on multiple cloud service providers. As shown in Fig. 1, the model consists of users, a service directory, and a collective group of cloud storage providers. The users are cloud clients who wish to store and access data in the clouds. A user can first interact with the service directory, which acts as a coordinator that can set up a cloud cluster with multiple cloud service providers and assign it to the user, and also store information regarding the addresses of all service providers in the a cloud cluster.

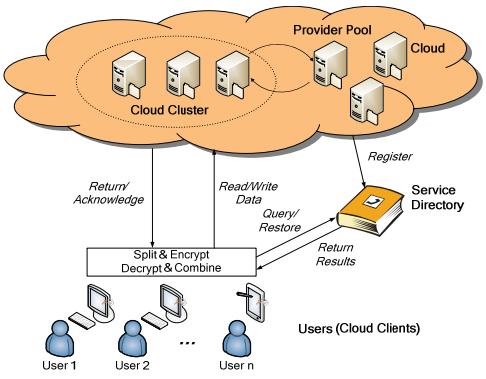

Figure 1. Architectural design of the information storage model

Once the client obtains the address information of the cloud providers from the service directory, it interacts with the cloud cluster, namely a collection of available service providers that can store and send data using predefined

protocols. Each set of data, composed of the user's sensitive information can be split into multiple pieces using a predefined security mechanism, and are stored into the cloud cluster after they are encrypted. The cloud providers in the cloud cluster have no knowledge of which cluster they belong to as well as which are the other members in the cloud cluster they are servicing. This means the cloud clusters are *virtual* clusters in the clouds, which are generated and exclusively managed by the service directory. Furthermore, the service directory has the capability to restore data when needed. When a service provider in a cloud cluster fails, the service directory can automatically restore the data using a predefined restoration algorithm, and replace the failed service provider with a new one from the *Provider Pool*.

The security mechanism defined in our model consists of two levels. In the first level, the information to be stored is split into multiple pieces using the *RAID 5* techniques with distributed parity [6] so that if a provider fails, the data stored collectively in the cluster would be recoverable. Note that the *RAID* technique uses block-level striping with distributed parity in a cluster of disk drives [7]. Due to data redundancy, when a disk drive fails, any subsequent reads can be calculated from the distributed parity, and the data in the failed drive can be restored. In our approach, we consider each cloud provider in a cloud cluster as a virtual disk drive; thus, our information storage model is fault tolerant upon the failure of any cloud provider in the cloud cluster, and the missing piece of data can be recovered from the distributed parity stored with the other cloud providers in the cloud cluster. Another advantage of our approach is, due to the distribution of data over multiple cloud providers, no cloud provider is able to calculate the original data because the providers have no knowledge of which the other members are in the cloud cluster.

In the second level of our security mechanism, encryption plays an important role. To ensure that providers do not have access to the underlying data that is being stored, symmetric key encryption can be used. A symmetric key is an encryption key that is used for both encryption and decryption of data and should be kept secret from all entities that do not have access to the data. A user with the needed access permission can utilize a symmetric key to encrypt a piece of data to be stored prior to sending it out to the cloud and to decrypt the data after it is retrieved from a cloud provider. When a read operation is performed, all pieces of information need to be decrypted after retrieved from the cloud providers in the cloud cluster, and then they are combined into the original information.

In order to correctly design the security mechanism, we develop a formal model of the secure and fault-tolerant information storage system in cloud computing, and verify some key properties of the model. We adopt colored Petri net formalism because it is a well-founded process modeling technique that has formal semantics to allow specification, design, verification, and simulation of a complex concurrent software system [8]. A Petri net is a directed, connected, and bipartite graph, in which each node is either a place or a transition. In a Petri net model, tokens are used to specify information or conditions in the places, and a transition can fire when there is at least one token in every input place of the transition. Colored Petri nets (CPN or CP-net) are an extension of ordinary Petri nets, which allow different values (represented by different colors) for the tokens. Colored Petri nets have a formal syntax and semantics that leads to compact and operational models of very complex systems for modular design and analysis. The major advantage of developing a CPN model of the information storage system is to provide a precise specification, and to ensure a correct design of the information storage system; therefore, design errors, such as a deadlock, can be avoided in the implemented system.

III. Formal Modeling Secure and Fault-Tolerant Cloud-Based Information Storage Archietcture

To make the model easy to comprehend, we utilize hierarchical CPN (HCPN), which allows using substitution transitions and input/output ports to represent a secondary Petri net in the hierarchy. In our design, we first provide the high-level model with its key components. Then we utilize HCPN to refine each component into a more complete Petri net. Since the architecture we proposed is most suitable for storing personal or confidential data, In the following sections, we present the HCPN model with an example of medical record online storage system, which consists of a service directory, a cloud cluster with three cloud providers, and two users (cloud clients), namely a patient and a doctor.

A. High-Level Petri Net Model

The HCPN model can be devloped using CPN Tools [9]. In Fig. 2, we present a high-level model that defines the key components, namely the *Doctor*, the *Patient*, the *Cloud*, and the *Directory*, as well as the communications among the components. The key components are defined as substitution transitions, denoted as double rectangles in the figure. The purpose of the communications among the patient, doctor, and cloud is to transfer and access a patient's medical record. The directory acts as a data coordinator between the users and the cloud. To simulate the cloud providers that are selected as members of a cloud cluster as well as the data being transferred between the users and the cloud providers, a `PROV` and a `MEDRECORD` colored token type are defined using the ML functional language integrated in CPN Tools as follows:

```
colset PROV = record
  prID: STRING *
  ready: BOOL *
  mrec: MEDRECORD;
```

```
colset MEDRECORD = record
  recID:STRING *
  data: STRING;
```

where `prID` is a provider ID, `ready` is a flag of a provider indicating whether the provider is functioning or failed, `mrec` is a medical record, `recID` is a record ID, and `data` is the medical data stored in the record.

The directory is responsible for initializing the cloud providers in a cloud cluster assigned to a user, replying queries from a user for providers' addresses, and processing restoration request upon the failure of a cloud provider in a cloud cluster. As shown in Fig. 2, the cloud cluster (denoted as the place "*Cluster Providers*") is initialized with three providers `"Pr1"`, `"Pr2"` and `"Pr3"` of type `PROV`, each of which is initialized with `initrec` that contains a blank medical record. Furthermore, the place "*Provider Pool*" is initialized with one spare cloud provider `"Pr4"`, which can be used to replace a failed cloud provider in the cloud cluster.

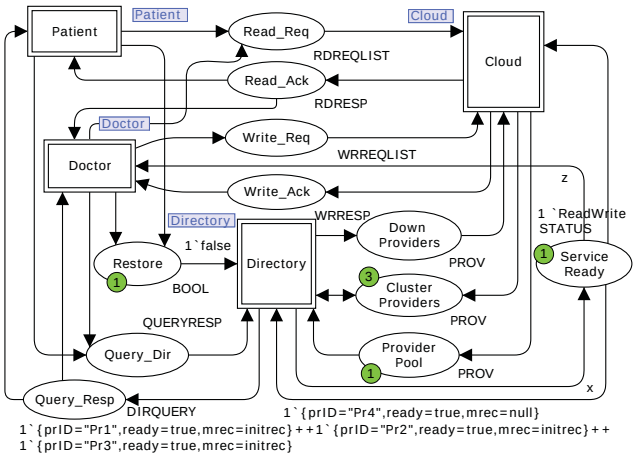

Figure 2. High-level CPN model of the cloud storage

A read request (`RDREQ`) and a write request (`WRREQ`) to a cloud provider can be defined as colored tokens as follows:

```
colset RDREQ = record
  clID:STRING *
  recID:STRING *
  prID:STRING;
```

```
colset WRREQ = record
  clID:STRING *
  mrec:MEDRECORD *
  prID:STRING;
```

where `clID` is a client ID. Note that in Fig. 2, `RDREQLIST` and `WRREQLIST` are defined as a list of read requests, and a list of write requests, repectively. Thus, our model allows accessing multiple pieces of information concurrently from the cloud providers participating in a cloud cluster.

After a read (write) requst has been processed, a read (write) response will be returned to the user, simulated as a token of type `RDRESP` (`WRRESP`) being deposited in place "*Read_Ack*" ("*Write_Ack*"). The colored token types `RDRESP` and `WRRESP` are defined as follows:

```
colset RDRESP = record
  clID:STRING *
  prID:STRING *
  mrec:MEDRECORD *
  success:BOOL;
```

```
colset WRRESP = record
  clID:STRING *
  prID:STRING *
  mrec:MEDRECORD *
  success:BOOL;
```

where the flag `success` indicates if a read request or a write request is successful or failed. In case a read or write request fails (i.e., a cloud provider is down), the user will change the token in place "*Restore*" from `false` to `true`, notifying the directory to start the restoration process for the cloud cluster.

B. Petri Net Model for the Directory Component

We now refine the *Directory* component (i.e., the *Directory* substitution transition in Fig. 2) into a CPN model as shown in Fig. 3. In the figure, the place "*Clust_Prov_List*" is intialized with a list of providers `["Pr1","Pr2","Pr3"]` due to the initial setting of the cloud cluster in place "*Cluster Providers*." When a patient client or a doctor client starts querying the directory for the addresses of the providers in its assigned cloud cluster, a query token will be placed by the client into place "*Query_Dir*." This enables the transition "*Provider Locations*." When it fires, it creates a token of `QUERYRESP` type in palce "*Query_Resp*," which attaches the provider information stored in place "*Clus_Prov_List*." Note that to simplify our CPN model, the provider information only consists of the provider IDs rather than the providers' actual endpoint addresses. Therefore, a service invocation to a cloud provider could be simulated by matching the cloud provider's ID rather than calling at its endpoint address. Since the place "*Query_Resp*" is an input port of the clients, the token becomes available to the client for further processing. On the other hand, if the "*Restore*" place contains a `true` token due to an access error experienced by a user, the "*Check Providers*" transition becomes enabled as long as the directory is not currently restoring the cloud cluster (denoted by a `false` token in place "*Init_Restore*") and there is a failed provider (i.e., its `ready` flag is set to `false`) in place "*Cluster Providers*." Once the transition fires, it places a `true` token into the "*Init_Restore*" place, signifying that a restoration process should take place. The firing also removes the failed provider from the "*Clust_Prov_List*" place and transfers the provider from the "*Cluster Providers*" place to the "*Down Provider*" place. When the restoration process starts, the "*Restore*" transition fires, and deposits a copy of the remaining two providers into the "*Restore Gather Info*" place. This enables the "*Calculate Replacement*" transition, and its firing simulates the calculation of the missing piece of data based on the distributed parity information, and results in the restored medical record being placed in the "*Replacement Record*" place. Note that for simplicity, the detailed procedure of the parity calculation is not modeled in Fig. 3.

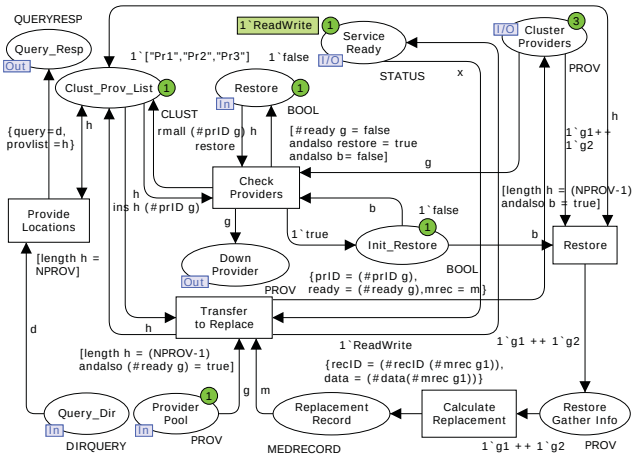

Figure 3. CPN model for the Directory component

Once the record has been restored, the "*Transfer to Replace*" transition becomes enabled, and its firing takes a provider from the "*Provider Pool*," initializes it with the restored medical record, updates provider list in the "*Clus_Prov_List*" place by adding the new provider into the list, and places the provider into the "*Cluster Providers*" place. This step completes the restoration process, with the required number of functioning providers allocated in the cloud cluster.

C. Petri Net Models for the Patient and Doctor Clients

A patient client should have the permisison to read its medical record. As shown in Fig. 4, a patient first requests the addresses of the cloud providers in the cloud cluster assigned to him, which is modeled by placing a `true` token in the

"*Query Directory*" place. With this token as well as the client ID of the user in the "*ClientID*" place, the "*Init_Query*" transition can fire, and its firing results in a `DIRQUERY` token to be placed in the "*Query_Dir*" output port.

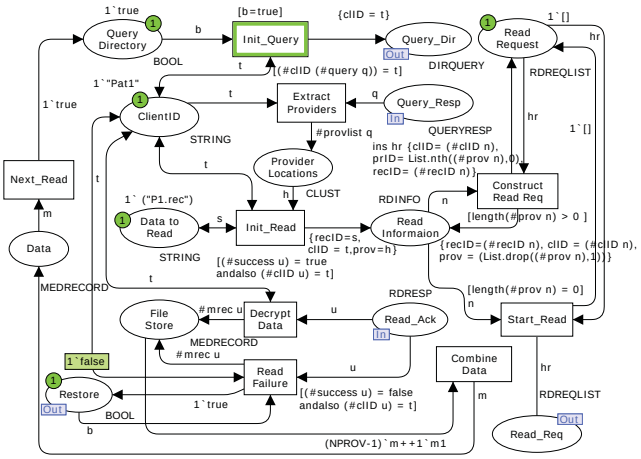

Figure 4. CPN model for the patient client

When a response from the directory is put into the "*Query_Resp*" input port, the providers' address information becomes available. This enables the "*Extract Providers*" transition, and the firing of the transition places a `CLUST` token in the "*Provider Locations*" place. The `CLUST` token type is defined as a list of providers as follows:

```
colset CLUST = list STRING with 0..3;
```

where the `with` clause specifies the minimum and maximum length of the list, and each item in the list contains the address of a provider (represented by its provider ID as a string for simplicity) that can be used by the client to communicate with the provider. To model a read operation, a token `"P1.rec"` is initialized in the "*Data to Read*" place, which is a record ID representing patient `P1`'s medical record. The firing of the "*Init_Read*" transition starts the read process, and places the record ID along with the provider information into the "*Read Information*" place. Note that in this model, we assume that there is only one record for each patient that can be matched with medical data stored on the providers. Now the "*Construct Read Req*" transition can fire once for each provider in the provider list, and creates a token of type `RDREQLIST` in the "*Read Request*" place, such that the multiple read requests in the list can be made concurrently to the cloud providers in the cloud cluster. This makes the associated providers in place "*Read Information*" being removed and enables the "*Start Read*" transition. When it fires, it transmits the `RDREQLIST` token to the "*Read_Req*" place, which is an input port to the cloud cluster. After the requests have been processed by the cloud providers, multiple tokens of type `RDRESP` will be deposited in place "*Read_Ack*." If a `RDRESP` token contains a `success` flag with a `true` value, it indicates that the read request has been completed successfully by the corresponding cloud provider. In this case, the piece of medical record is extracted from the token and placed in the file store after bieng decrypted. Once all pieces of the medical record are successfully decrypted, the "*Combine Data*" transition becomes enabled and can fire. The firing simulates the process of generating the original medical record by recombining the *RAID* data slices retrieved from the cloud providers. If one of the providers returns a token with the `success` flag set to `false`, a read failure occurs for the cloud provider. In this case, the "*Read Fail*" transition becomes enabled. Once it fires, it changes the token in place "*Restore*" from `false` to `true`, signifying the directory to initiate a "restore" operation.

The CPN model for the doctor client that replaces the `Doctor` substitution transition of the high-level model is similar to the one for the patient client, but a doctor client should also have the privilege to write data into the clouds. Due to page limits, we do not show such a CPN model here.

D. Petri Net Model for the Cloud Component

Finally, we refine the *Cloud* substitution transition of the high-level model into a CPN model as shown in Fig. 5, where the cloud providers are represented as colored tokens of type `PROV`. The clouds can accept either "read" or "write" requests from the clients, namely the patient and the doctor. Upon receiving the requests, cloud providers invoke corresponding cloud services by matching their IDs in the cloud cluster, and return responses to the clients. In addition, the cloud providers in a cloud cluster are also responsible for providing their data to the service directory on demand in a case that a restoration process is initiated when a read or write request fails due to the failure of a cloud provider in the cloud cluster.

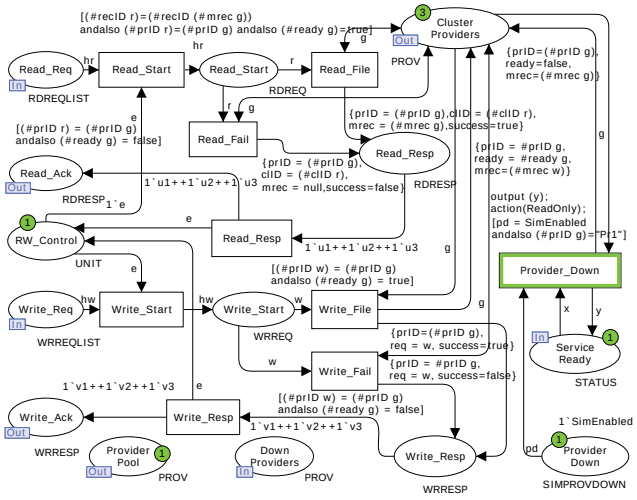

Figure 5. CPN model for the cloud component

In this model, the "*Cluster Providers*" place is shared with the directory, where the `PROV` tokens in the place represent the providers selected to constitute the cloud cluster. In addition, the "*Provider Pool*," "*Down Providers*," and "*Service Ready*" places are also shared places in the directory model. The "*Provider Pool*" acts as a holding place for available providers identified by the directory. The "*Down Providers*" place contains the providers that are down and deemed needing replacement. Finally, the "*Service Ready*" place acts as an input place to the cloud for simulation purposes only. In our current model, we only consider a maximum of one cloud provider

going down at a time. This is a reasonable assumption because cloud providers should be somewhat reliable. In order to satisfy this constraint in the model, the "*Provider Down*" place is connected to the "*Provider_Down*" transition, which allows the "*Provider_Down*" transition to fire once.

When a client makes a "read" request, a `RDREQLIST` token with a list of `RDREQ` requests will be deposited into place "*Read_Req*." This enables the "*Read_Start*" transition as long as the "*RW_Control*" place contains a unit token, which ensures "read" and "write" actions are mutual exclusive. When the "*Read_Start*" transition fires, it splits the `RDREQLIST` token into singular `RDREQ` tokens, places them into place "*Read_Start*," and removes the unit token from the "*RW_Control*" place. The "*Read_File*" transition then examines each of the `RDREQ` tokens, matches it with its respective cloud provider, and fires as long as the `success` flag of the corresponding `PROV` token in place "*Cluster Provider*" is `true`. The following ML transition guard code accomplishes this task:

```
[(#recID r) = (#recID (#mrec g)) andalso
 (#prID r) = (#prID g) andalso (#ready g)=true]
```

where `g` represents a cloud provider that is a member of the cloud cluster and `r` represents a "read" request. The guard selects the correct provider by comparing the provider ID in the request `(#prID r)` with that of a provider from the cluster`(#prID g)`, matches the medical record ID, and makes sure that the cloud provider is functioning, i.e., its `ready` flag is set to `true`. If all conditions are met, the transition can fire, and the firing of the transition creates a `RDRESP` token and deposits it into the "*Read_Resp*" place. On the other hand, if a "read" request fails due to the corresponding provider being not ready (i.e., its `ready` flag is set to `false`), the "*Read_Fail*" transition can fire, and its firing sends a `RDRESP` token with a blank medical record and a `success` flag set to `false` to the "*Read_Resp*" place. Once all three tokens are in the "*Read_Resp*" place, the "*Read_Resp*" transition may fire. The firing of the transition returns a unit token to the "*RW_Control*" place and places the `RDRESP` tokens into the "*Read_Ack*" port, available for the clients to digest.

A "write" request follows an almost identical path through the model. When the doctor places a `WRREQLIST` token into the "*Write_Req*" port, the "*Write_Start*" transition becomes enabled, and the firing of the transition places the individual `WRREQ` tokens into place "*Write_Start*." With the tokens in this place, the "*Write_File*" transition can fire as long as the `ready` flag of some `PROV` token in place "*Cluster Providers*" is `true`. The firing of the transition replaces the medical record stored in the `PROV` token with the replacement record, and also constructs a `WRRESP` token and places it in the "*Write_Resp*" place. On the other hand, if the `ready` flag of a provider is set to `false`, the "*Write_Fail*" transition may fire. In this case, the medical record is not altered, and a `WRRESP` token with the `success` flag set to `false` will be deposited in place "*Write_Resp*." Once all three `WRRESP` tokens are in the "*Write_Resp*" place, the "*Write_Resp*" transition can fire, and its firing returns a unit token back to the "*RW_Control*" place and deposits the `WRRESP` tokens in the "*Write_Ack*" place, being available for the client to process.

A restoration process can be simulated in the cloud model by setting the `SIMPROVDOWN` token in place "*Provider Down*" to `SimEnabled`. When the "*Provider_Down*" transition fires, it randomly selects a provider from the place "*Cluster Providers*" and sets the `ready` flag of the provider to `false`. This step simulates the failure of a cloud provider in the cloud cluster. Furthermore, the firing of the transition also sets the `STATUS` token in place "*Service Ready*" to `ReadOnly`, which disables the transition for writing in the CPN model for the doctor patient. The doctor patient will be allowed to write again only after the `STATUS` token in place "*Service Ready*" is changed back to `ReadWrite`. Meanwhile, when either a patient or a doctor client experiences an access error to a failed cloud provider, a restoration process will be initiated by the client. Communication with the directory for a "restore" operation is done through the shared port "*Cluster Providers*." This port, containing the `PROV` tokens of the providers who make up the cluster, allows the directory direct access to the `PROV` state when required. When the restoration process completes, the failed cloud provider in place "*Cluster Providers*" will be replaced by a new one taken from the "*Cluster Pool*."

IV. Formal Analysis of the CPN-Based Model

In addition to providing an accurate model for our proposed security mechanisms for cloud-based information storage, building a formal design model also has the advantage of ensuring a correct design through state space analysis. Utilizing the CPN Tools, a formal analysis of the CPN model can be performed to verify if the model meets certain system requirements. Typically, the model we developed should be live, bounded, and deadlock-free. When we use the CPN Tools to calculate the state space and analyze its major behavioral properties, the CPN Tools produce the following results:

```
Statistics                 Liveness Properties
-----------------------    --------------------------
  State Space                Dead Markings
     Nodes:  154908             99 [44684,44683,44682,
     Arcs:   571408             44510,44509,...]
     Secs:   1832             Dead Transition Instances
     Status: Full                None
  Scc Graph                  Live Transition Instances
     Nodes:  90616               None
     Arcs:   431478
     Secs:   37
```

The analysis shows that the state space contains dead markings, thus the model we developed must contain deadlocks. By tracing the firing sequence for the deadlock states as we did in our previous work [10], we found a subtle design error. The error is due to the removal of the failed cloud provider from the place "*Cluster Provider*" in the CPN model for the *Directory* component, which occurs when the transition "*Check Providers*" fires. However, some "read" request in place "*Read_Req*" of the CPN model for the *Cloud* component would require communicating with a removed cloud provider if the "read" request was created before the cloud provider fails. Since there is no matched cloud provider in the "*Cluster Provider*" place of the *Directory* model, the system may enter a deadlock state. The easiest way to fix this problem is to allow the failed cloud provider to stay in the "*Cluster Provider*" place. This would allow the "*Read_Fail*"

transition to fire, and return a "read" error to the client. After we add a new arc from the transition "*Check Providers*" to the place "*Cluster Provider*" in the *Directory* model, the CPN Tools now produce the following results:

```
Statistics                Liveness Properties
-----------------------   -------------------------
  State Space               Dead Markings
     Nodes:  204267            None
     Arcs:   880021         Dead Transition Instances
     Secs:   4599              None
     Status: Full           Live Transition Instances
  Scc Graph                    Cloud'Read_File 1
     Nodes:  133063            Cloud'Read_Resp 1
     Arcs:   726216            Cloud'Read_Start 1
     Secs:   242               ...
```

```
Boundedness Properties
-------------------------------------------------
  Place                          Upper      Lower
  Cloud'Provider_Down              1          0
  Cloud'RW_Control                 1          0
  Cloud'Read_Resp                  3          0
  Cloud'Read_Start                 3          0
  Cloud'Write_Resp                 3          0
  Cloud'Write_Start                3          0
  Directory'Clust_Prov_List        1          1
  Directory'Init_Restore           1          0
  Directory'Replacement_Record     1          0
  Directory'Restore_Gather_Info    2          0
  High_Level'Cluster_Providers     4          3
  High_Level'Down_Providers        1          0
  High_Level'Provider_Pool         1          0
  High_Level'Query_Dir             2          0
  High_Level'Query_Resp            2          0
  High_Level'Read_Ack              6          0
  High_Level'Read_Req              2          0
  High_Level'Restore               1          1
  High_Level'Service_Ready         1          1
  High_Level'Write_Ack             3          0
  High_Level'Write_Req             1          0
  ...
```

The analysis shows that our modified net model is deadlock free, and all transitions except those related to the restoration process are live. Note that in our simulation, we allow the "*Provider_Down*" transition in the *Cloud* model can fire only once. The analysis also shows that our net model is bounded. We notice that the upper bound of the place "*Cluster_Providers*" in the high-level model is 4 rather than 3. This is because a failed cloud provider will be kept in the cloud cluster after the service directory restores the cloud cluster by adding a replacement cloud provider into the cluster. A more sophisticated model that allows a failed cloud provider to be removed from the cloud cluster, and also allows more than one patient and more than one doctor to access cloud clusters with shared cloud providers is envisioned as a future, and more ambitious research plan.

V. Conclusions and Future Work

Cloud computing is quickly becoming a widely adopted platform to allow for complex computational nodes and storage clusters without any of the difficulties and cost associated with configuration and maintenance. There are, however, major legitimate concerns from enterprises and sensitive data holders related to offsite storage of personal or mission critical data. Studies show that given the current state of cloud computing, enterprises are very concerned with unresolved issues related to security, trust, and management in the cloud. For a majority of these enterprises, this is also the main reason why they have not yet adopted cloud computing into their infrastructure. In this paper, we introduce a cloud-based information storage model that takes into account the fact that cloud providers may experience outages, data breaches, and exploitations. We cope with these issues by developing a distributed cloud-based security mechanism. We then utilize hierarchical colored Petri nets to formally model and analyze our concurrent security model. The verification results show that the model we developed is live, bounded and deadlock-free.

For future work, we plan to develop a more sophisticated model that allows more clients to access the cloud clusters with shared cloud providers, and demonstrate how to cope with the state explosion problem using the net reduction approach. Meanwhile, we will try to implement a prototype cloud-based information storage system with an improved distributed parity algorithm that may strengthen the security mechanism by preventing potential provider collusion to obtain information stored in a cloud cluster. Finally, the model can be further improved if we allow the service directory to autonomously detect failures, drops in quality of service (QoS), and anomalies of the cloud providers, and react accordingly.

References

[1] GAO, "Information Security: Additional Guidance Needed to Address Cloud Computing Concerns," *United States Government Accountability Office (GAO)*, October 6, 2011, Retrieved on December 18, 2011 from http://www.gao.gov/new.items/d12130t.pdf

[2] AWS, "Creating HIPAA-Compliant Medical Data Applications with AWS," *Amazon Web Services (AWS)*, Amazon, April 2009, Retrieved on August 22, 2010, from http://awsmedia.s3.amazonaws.com/AWS_ HIPAA _Whitepaper_Final.pdf.

[3] M. T. Goodrich, R. Tamassia, and D. Yao, "Notarized Federated ID Management and Authentication," *Journal of Computer Security*, Vol. 16, No. 4, December 2008, pp. 399-418.

[4] A. C. Weaver, "Enforcing Distributed Data Security via Web Services," In *Proceedings of the IEEE International Workshop on Factory Communication Systems (WFCS)*, Vienna, Austria, September 22-24, 2004, pp. 397-402.

[5] N. Santos, K. Gummadi, and R. Rodrigues, "Towards Trusted Cloud Computing," In *Proceedings of the Workshop on Hot Topics in Cloud Computing (HotCloud09)*, San Diego, CA, USA, June 15, 2009.

[6] A. Thomasian and J. Menon, "RAID 5 Performance with Distributed Sparing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 6, June 1997, pp. 640-657.

[7] D. A. Patterson, P. Chen, G. Gibson, and R. H. Katz, "Introduction to Redundant Arrays of Inexpensive Disks (RAID)," *COMPCPN Spring'89, Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers*, Feb. 27 - March 3, 1989, San Francisco, CA , USA, pp. 112-117.

[8] K. Jensen, *Colored Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Vol. I: Basic Concepts, EATCS Monographs on Theoretical Computer Science, New York Springer-Verlag, 1992.

[9] A. V. Ratzer, L. Wells, H. M. Lasen, M. Laursen, J. F. Qvortrup, et al., "CPN Tools for editing, simulating and analyzing colored Petri nets," In *Proceedings of the 24th International Conference on Application and Theory of Petri Nets*, Eindhoven, Netherlands, Jun. 2003, pp. 450-462.

[10] H. Xu, M. Ayachit, and A. Reddyreddy, "Formal Modeling and Analysis of XML Firewall for Service-Oriented Systems," *International Journal of Security and Networks (IJSN)*, Vol. 3, No. 3, 2008, pp. 147-160.