Cloud Service Reliability: Modeling and Analysis

Yuan-Shun Dai *[a c], Bo Yang[b], Jack Dongarra[a], Gewei Zhang[c]

[a] Innovative Computing Laboratory, Department of Electrical Engineering & Computer Science, University of Tennessee, Knoxville, TN, USA

[b] Collaborative Autonomic Computing Laboratory, School of Computer Science University of Electronic Science and Technology of China, Chengdu, China

[c] Department of Industrial and Information Engineering, University of Tennessee, Knoxville, TN, USA

Abstract

Cloud computing is a recently developed new technology for complex systems with massive-scale service sharing, which is different from the resource sharing of the grid computing systems. Cloud reliability analysis and modeling are not easy tasks because of the complexity and large scale of the system. This paper systematically analyzes cloud computing and models the reliability of the cloud services. Various types of failures are interleaved in the cloud computing environment, such as overflow failure, timeout failure, resource missing failure, network failure, hardware failure, software failure, and database failure. This paper investigates all of them to achieve a comprehensive picture about cloud service reliability, and models those failures in a holistic manner using Markov models, Queuing Theory and Graph Theory. In accordance with the proposed model, a new evaluation algorithm is further developed in this paper integrating the Bayesian approaches together with the Graph Theory.

Keyword

Cloud computing, Reliability modeling, Graph theory, Queuing theory, Bayesian analysis

1. Introduction

Cloud computing enables the massive-scale service sharing, which allows users to access technology-enabled services without knowledge of, expertise with, or control over the technology infrastructure that supports them. Cloud computing is different from but related with grid computing, utility computing and transparent computing. Grid computing [1] is a form of distributed computing whereby a "super and virtual computer" composed of a cluster of networked, loosely-coupled computers acts in concert to perform very large tasks. Utility computing [2] is the packaging of computing resources, such as computation and storage, as a metered service similar to a traditional public utility such as electricity. Transparent computing [3] means complex back-end services are transparent to users who only see a simple and easy-to-use front-end interface. The cloud computing deployments are today powered by grids, having transparent characteristics and billed like utilities; but cloud computing is rather a natural next step from the grid-utility-transparent model. Based on this model, the cloud computing can rather realize the service sharing than only the resource sharing coined by grid computing.

The cloud computing is thus more service-oriented than resource-oriented. Dai *et al.* [4] has already mentioned that the users do not care too much about the resources of the grid system but are more concerned with the services they are using. Hence, the function of service sharing enabled by cloud computing will be more interesting to general users than the resources sharing of the grid computing. A variety of cloud services are provided by the cloud system. The cloud system could become very large even all over the whole Internet. Users can request cloud services from any corner of the world. Some examples of commercial cloud services include Amazon EC2 [5], Xen [6], Google Cloud [7], IBM Cloud [8], and Microsoft Cloud [9].

The reliability of the cloud computing is very critical but hard to analyze due to its characteristics of massive-scale service sharing, wide-area network, heterogeneous software/hardware components and complicated interactions among them. Hence, the reliability models for pure software/hardware or conventional networks [10-11] cannot be simply applied to study the cloud reliability.

Therefore, this paper first presents an innovative reliability model for cloud computing. The cloud reliability model is service oriented and hierarchical, which is tractable and effective in addressing such a large and complex system. This new model comprehensively considers various types of failures that have significant influences on the success/failure of cloud services, including overflow, timeout, data resource missing, computing resource missing, software failure, database failure, hardware failure, and network failure.

The remaining of this paper is organized as follows. Section 2 presents a general architecture of the cloud computing system and makes a thorough analysis of the cloud services. Section 3 builds a holistic model for cloud service reliability and presents a new evaluation algorithm. Section 4 concludes this paper and discusses the future research.

2. Cloud Computing System and Failure Analysis

Cloud computing is distinguished from conventional distributed computing by its focus on massive-scale service sharing. The characteristics of the cloud computing are described in subsection 2.1, and then various failures in a cloud service are analyzed in subsection 2.2.

2.1. Description of the cloud computing

We are developing a cloud computing system in the VGrADS (Virtual Grid Application Development Software) project sponsored by National Science Foundation (NSF). This system has already been collaborating and integrated with Amazon EC2 [5]. The architecture of our cloud service system is depicted in Fig. 1, which is also a typical representation of most present or future cloud service systems. There is a cloud management system (CMS) which is composed by a set of servers (either centralized or distributed). The CMS mainly fulfills four different functions as shown in Fig. 1: 1) To manage a request queue that receives job requests from different users for cloud services; 2) To manage computing resources (such as PCs, Clusters, Supercomputers, etc.) all over the Internet; 3) To manage data resources (such as Databases, Publicized Information, URL contents, etc.) all over the Internet; and 4) To schedule a request and divide it into different subtasks and assign the subtasks to different computing resources that may access different data resources over the Internet.

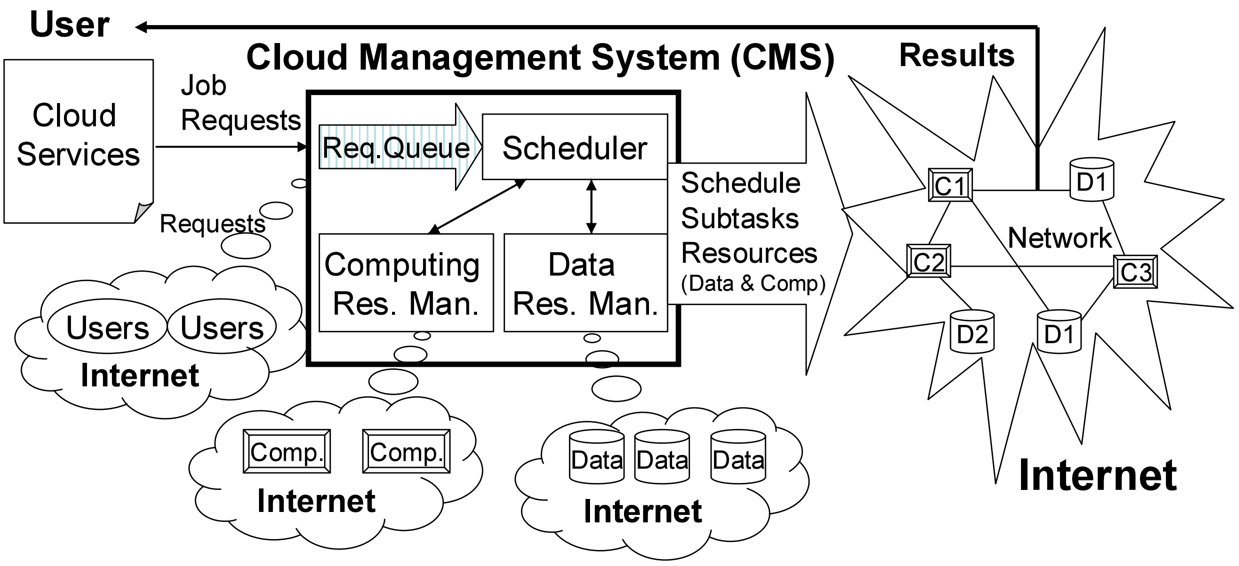

Fig. 1. Cloud Service System.

When a user requests a certain given cloud service, we apply a workflow to describe and manage the cloud service [12]. Fig. 2 depicts a workflow template of a service that includes four different subtasks (S1, S2, S3, S4) and their interrelationship (data dependency), e.g. S3 needs the inputs that result from S1 and S2. It also shows the required data resources that the subtasks need to access, e.g., S1 needs to access data resource D1 when running, S2 needs D2 and D3, and S4 needs D4, but S3 needs nothing. With the given workflow of a cloud service, the scheduler in the CMS can assign these subtasks to different computing resources while allocating the data resources, as shown in Fig. 2, e.g., the computing resource C1 is assigned two subtasks, S1 and S3, to run, C5 is a data resource offering data D2, D3 and D4, and C3 is both computing resource and data resource to run subtask S2 while offering data D1 and D3. After the computing resources and data resources receive the commands/subtasks from the CMS, they form a network according to the connectivity or accessibility, e.g. C3 is directly connected with C5, but cannot directly communicate with C4 due to the connectivity (e.g. computers C3 and C4 may be both behind routers that translate their original IP addresses so that they cannot directly build the TCI/IP connection, or they do not have access to each other [13]).

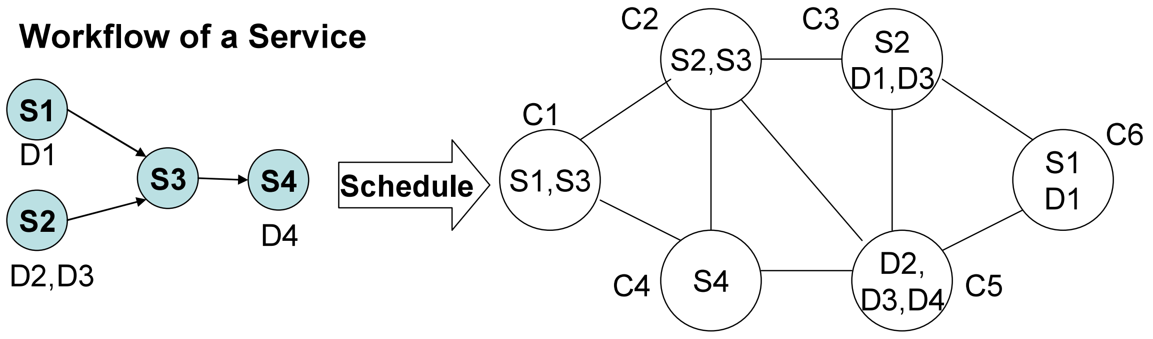

Fig. 2. Workflow of a Cloud Service and Scheduling

The cloud network shown in Fig. 2 can be very large, and each link in Fig. 2 is actually a virtual link that may go through many components (routers/cables/optical fibers/machines) over a long distance. Thus, the computing resources will work together via the network to run the subtasks while accessing necessary data from the data resources. When the job is finished, the results will return to the user who requests this service, as shown in Fig. 1.

2.2. Failure Analysis of Cloud Service

As Fig. 1 and Fig. 2 show, there are a variety of types of failures that may affect the success/reliability of a cloud service, including Overflow, Timeout, Data resource missing, Computing resource missing, Software failure, Database failure, Hardware failure, and Network failure. We analyze these failures in more details.

- *Overflow*: The request queue should have a limitation on the maximal number of requests waiting in the queue. Otherwise, new requests have to wait for too long a time in the queue, which could make the Timeout failures much more dominant. Therefore, if the queue is full when a new job request arrives, it is simply dropped and the user is unable to get service, which is called an overflow failure.
- *Timeout*: The cloud service usually has its due time set by the user or the service monitor. If the waiting time of the request in the queue is over the due time, the Timeout failure occurs, see e.g. [14]. As a result, those timeout requests will be dropped from the queue so that not to affect other following requests.
- *Data resource missing*: In CMS, the data resource manager (DRM) registers all data resources. However, it is possible that some previously registered data are removed but

the DRM is not updated. As a result, if those data resources are assigned in a certain job request, they will cause the data resource missing failure.

- *Computing resource missing:* Similarly to the above data resource miss, the computing resource missing may also occur, such as PC turns off without notifying the CMS.
- *Software failure*: The subtasks are actually software programs running on different computing resources, which contain software faults, see e.g. [15].
- *Database failure:* The database that stores the required data resources may also fail, causing that the subtasks when running cannot access the required data.
- *Hardware failure:* The computing resources and data resources in general have hardware (such as computers or servers) which may also encounter hardware failures.
- *Network failure:* When subtasks access remote data, the communication channels may be broken either physically or logically, which causes the network failure, especially for those long time transmissions of large datasets, see e.g. [16].

The model for cloud computing reliability has to consider all types of these failures, which would be very complicated and existing reliability models cannot address all of these concerns in a holistic manner although each single topic has been studied.

Moreover, these different types of failures are actually correlated with one another (i.e., not independent) in a cloud service which exhibits another reason why the cloud reliability model cannot simply utilize any one single existing model in each individual topic (such as software reliability, hardware reliability, or network reliability). For example, failures of schedulers may increase the waiting time, which could affect the timeout and the overflow failures; a large queue limit may reduce the probability of overflow failure but may increase that of the timeout failure; a database failure may make a software unable to be finished due to lack of necessary data; network failures may block the required communications among software programs to get necessary inputs from others. With such correlations, it is obvious that a new holistic model has to be developed for cloud reliability.

3. Cloud Service Reliability Modeling and Evaluation

In this section, we develop a holistic model for *Cloud Service Reliability*, which is defined as the probability that a cloud service under consideration can be successfully completed for a user in a specified period of time. In particular, this requires that the job request be successfully served by the schedulers in time, the set of subtasks contained by the service be completed, the computing/data resources required by the subtasks be available; and the network be operational during the communications.

From the definition of cloud service reliability, it is clear that all types of failures we have discussed in section 2 will more or less affect this probability to provide a successful service. We classify the above failures in two groups:

1. *Request Stage Failures:* Overflow and Timeout.
2. *Execution Stage Failures:* Data resource missing, Computing resource missing, Software failure, Database failure, Hardware failure, and Network failure.

The failures in Group 1 may occur before the job request is successfully assigned to computing/data resources; on the other hand, the failures in Group 2 may occur after the job request has been successfully assigned and during the execution of subtasks. Therefore, the two groups of failures could be deemed as independent. Nevertheless, failures within each group are strongly correlated. In summary, the modeling of cloud service reliability can be separated in two parts: modeling of Request Stage Reliability and modeling of Execution Stage Reliability.

3.1. Request Stage Reliabiliy

This request stage contains two types of failures: overflow and timeout. The *due time* for a specific service is the allowed time spent from the submission of the job request to the completion of the job. The due time can be set by the user or by the service monitor. If a job request is not served by a scheduler before the due time, it will be dropped. The dropping rate is denoted by μ_d.

Suppose the capacity of the request queue is N (the maximal number of requests in the queue). We assume that the arrival of submissions of job requests follow a Poisson process with the arrival rate of λ_a.

Usually, there are multiple schedule servers to serve the requests. These schedule servers are usually homogeneous with similar structures, schemes and equipments. Here, we assume a total of S homogenous schedule servers are running simultaneously to serve the requests. The service time to complete one request by each schedule server is assumed to be an exponentially distributed quantity with parameter μ_r. Thus, such process can be modeled by a Markov process as depicted by Fig. 3, in which state *n* (*n*=0,1,…,*N*) represents the number of requests in the queue.

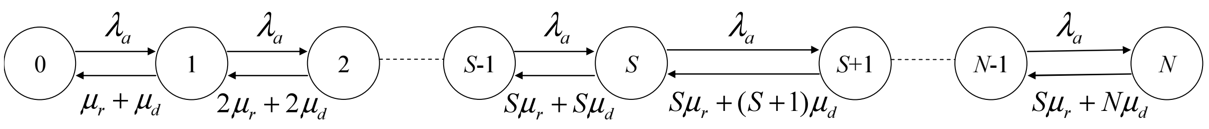

Fig. 3. Markov model for the request queue.

In Fig. 3, the transition rate from state *n* to state *n*+1 is λ_a. At state *N*, the arrival of a new request will make the request queue overflow, so the request is dropped and the queue still stays at state *N*. The service rate of a request by a schedule server is μ_r. If $n \le S$, then *n* requests can be immediately served by the *S* schedule servers, so the departure rate of any one request is equal to $n\mu_r$. If $n > S$, only *S* requests are being simultaneously served by schedule servers, so the departure rate is $S\mu_r$. The dropping rate for any one request in the queue to reach its due time is $n\mu_d$ (*n*=1,2,…,*N*).

Denote by q_n the steady probability for the system to stay at state *n* (*n*=0,1,…,*N*). It is easy to derive q_n by solving the following Chapman-Kolmogorov equations:

$$\lambda_a q_0 = \mu_r q_1 \tag{2}$$

$$\left(n\mu_r + \sum_{x=1}^{N-n} p(x)\lambda_a\right) q_n = (n+1)\mu_r q_{n+1} + \sum_{y=0}^{n-1} p(n-y)\lambda_a q_y \qquad (n=1,\ldots,S\text{-}1) \tag{3}$$

$$\left(S\mu_r + \sum_{x=1}^{N-n} p(x)\lambda_a\right) q_n = S\mu_r q_{n+1} + \sum_{y=0}^{n-1} p(n-y)\lambda_a q_y \qquad (n=S,\ldots,N\text{-}1) \tag{4}$$

$$S\mu_r q_N = \sum_{y=0}^{N-1} p(N-y)\lambda_a q_y \tag{5}$$

And

$$\sum_{n=0}^{N} q_n = 1 \tag{6}$$

The probability for the overflow failure NOT to occur is thus

$$R_{overflow} = \sum_{n=0}^{N-1} q_n , \tag{7}$$

where q_n (*n*=0,1,…,*N*) can be obtained by solving equations (2)-(6).

To study the timeout failure, suppose the current length of the request queue is *n* (*n*=0,1,…,*N*-1) when the new service request under consideration arrives. The probability density function (p.d.f.) of waiting time to complete the *n* requests by *S* schedule servers is

$$f_n(t) = S\mu_r e^{-S\mu_r t}\frac{(S\mu_r t)^{n-S}}{(n-S)!},\ t \geq 0 \text{ and } n \geq S. \tag{8}$$

If the waiting time is longer than the due time T_d, the timeout failure occurs. Therefore, the probability for the waiting time in completing the *n*+1 requests to be less than T_d is

$$\Pr\{t < T_d\} = \int_0^{T_d} f_n(t)dt \qquad n \geq S \tag{9}$$

If $n < S$, then the new request that has arrived can be immediately served without any waiting time. Therefore, the probability for the timeout and overflow failures NOT to occur (i.e. the Request Stage is reliable) is

$$R_{request} = \sum_{n=0}^{S-1} q_n + \sum_{n=S}^{N-1} q_n \int_0^{T_d} f_n(t)dt \tag{10}$$

where $f_n(t)$ can be obtained by (8). The summation in (10) between $[0, N-1]$ contains a condition that the overflow failure not to occur as analyzed by the (7). Thus, in (10) $R_{request}$ represents the probability without timeout or overflow failures.

3.2. Execution Stage

3.2.1. A New Model

To address various types of failures during the execution of a cloud service, we propose a new model here. All types of execution stage failures are integrated in this new model, as illustrated by a graph model in Fig. 4.

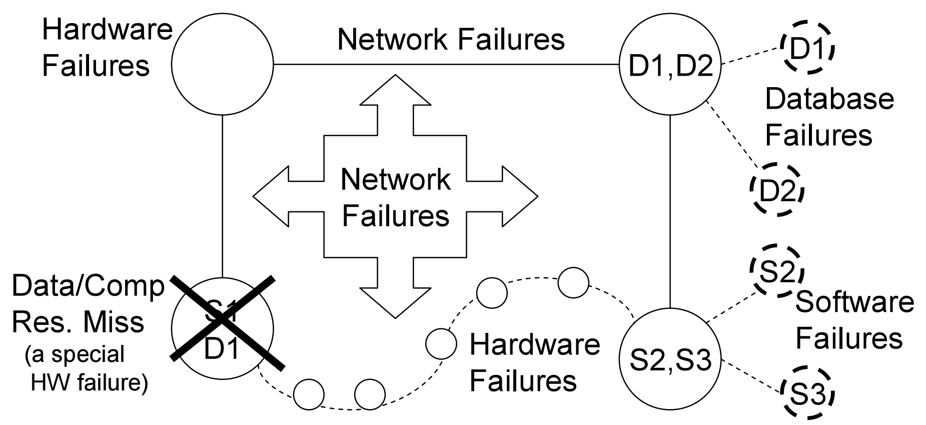

Fig. 4. A graph model integrating different types of failures at the execution stage.

In this model, *hardware* (such as a computer) is represented by a solid-line node, so the characteristics regarding the hardware (such as hardware failures, processing speed, etc.) can be associated with the node.

The link of the *network* is represented by a solid line which represents a communication channel between two nodes, so the characters of the channel (such as link failure, bandwidths, etc.) can be associated with the link.

Hardware may contain database or software required by the cloud service, so we suggest using *virtual nodes* to represent database or software programs, which are drawn as dashed-line circles. The idea of virtual nodes is different from previous graph models for distributed computing systems [17]. Those models only exhibit the software/database inside the hardware node, which actually fits the physical structure (e.g. software does run inside the computer hardware), but such physical representations could not represent the heterogeneity of hardware/software/databases so these models only used the node property to incorporate all different characteristics. However, in cloud computing the heterogeneity is significant including various kinds of resources, so these resources should be treated respectively. The virtual nodes making physical structure inside-out can fulfill this requirement. As a result, the characteristics with respect to the database (such as size of data uploaded/downloaded, database failures, etc.) and to the software (such as software failures, and the running time of the software) can be associated with different virtual nodes without interfering the characteristics of hardware (the solid-line node). This virtual link (dashed line) connects different virtual nodes to their hosted hardware node. The virtual structure in Fig. 4, when exhibiting the heterogeneity, can also show the failure correlation to accommodate to the practice better, e.g. if the hardware fails, then all those virtual nodes (components inside this hardware) are isolated from outside, which means unavailable at the same time to other external components.

Finally, this graph model can also address the data/computing resources missing. Once the missing resources are included by the cloud scheduler by mistakes, we can address the missing in another way, i.e., the resource fails at the beginning of the execution of the cloud service. Thereby, the missing of resources can be incorporated in the hardware property, as a special type of hardware failure.

In summary, the new graph model to be built as per the above methodology can well address those different failures in a holistic manner for a given cloud service during the execution stage.

3.2.2. Parameters

In accordance with the new model as depicted by Fig. 4, the parameters with respect to different components are discussed here, which will be used in the proposed evaluation algorithm.

For the *i*:th hardware node ($i = 1,2,...,H$), denote by ps_i its *Processing Speed*, e.g. in MIPS (Million Instructions per Second). For the *j*:th data resource ($j = 1,2,...,J$), denote by sd_j the *Amount of Data* downloaded/uploaded by remote software programs, e.g., in MB (Mega Bytes). For software (such as a software program to complete a subtask), denote by wp_k ($k = 1,2,...,K$) the *Workload* of the *k*:th software program, e.g. in NoI (Number of Instructions) to be executed. Denote by sd_{ij} ($i = 1,2,...,J$, $j = 1,2,...,J$, $i \neq j$) the *Amount of Data* exchanged between the *i*:th subtask and the *j*:th subtask. Denote by bw_m ($m = 1,2,...,M$) the *Bandwidth* of the *m*:th communication link, e.g. in bps (bit per second).

Any of the elements of hardware/database/software/links may encounter failures. The *failure rate* [11] is another parameter of interest. As explained by [18], in the operational phase of software, there will be no modifications made on the software source code, thus the software failure rate is a constant. For electronic hardware, a constant failure rate is normally observed in the operational phase as well. We thus denote by $\lambda(element_n)$ the failure rate of the *n*:th element. Therefore, the reliability of each individual element can be derived as

$$R(element_n) = \exp\{-\lambda(element_n) \cdot T_w(element_n)\}, \tag{11}$$

where $T_w(element_n)$ denotes the length of working time of the n:th element in a cloud service, which can be derived, respectively, as follows.

The time that the k:th software program is running on the i:th machine is

$$T_w(Software) = \frac{\text{Software Workload}}{\text{Processing Speed}} = \frac{wp_k}{ps_i} \tag{12}$$

The time that the m:th communication link is transmitting data is

$$T_w(Communication) = \frac{\text{Amount of Data}}{\text{Bandwidth}} = \frac{sd_{ij}}{bw_m} \tag{13}$$

The total working time for a hardware element has two parts: running software and transmitting data, thus

$$T_w(Hardware) = \sum_{Hardware} T_w(Software) + \sum_{Hardware} T_w(Communication) \tag{14}$$

which means the summation of the execution time of all software programs running on this hardware and the communication time of all channels going through this hardware.

The working time for a data source can be calculated as the summation of all communication times that access the data on the data source.

$$T_w(DataSource) = \sum_{Data} T_w(Communication) \tag{15}$$

With the working time derived by equations (12)-(15), the reliability of individual element can be obtained from (11), which is more realistic and practical than other conventional methods [17] assuming the reliabilities of elements (nodes and links) are constant, (e.g. a node is always 90% reliable, regardless of how long it works). In fact, the reliability of individual element is affected by various conditions such as failure rate, amount of data, bandwidth, operation time, etc.

3.2.3. New Evaluation Algorithm

Though the new graph model and the parameters of elements are more realistic and practical, they also make the evaluation of overall reliability much more complicated so that the existing algorithms [17] could not be directly applied here. For instance, those conventional algorithms have one or some of the following assumptions that are not applicable to evaluate the reliability given the above new model: 1) the network topology is made up of physical nodes/links without considering the virtual nodes/links; 2) the operational probabilities (reliabilities) of nodes or links are constant; 3) only hardware failures of links and processors are considered without taking into account the software, data and resource failures.

Therefore, we further present a new algorithm for evaluating the overall cloud service reliability considering all different factors during the execution stage given the new graph model and the above parameters. The new evaluation algorithm based on Graph theory and Bayesian theorem is presented to derive the reliability, as follows.

A. Minimal Subtask Spanning Tree (*MSST*)

The set of all nodes and links involved in completing a specific subtask form a *Subtask Spanning Tree* (*SST*). This SST can be considered to be a combination of several minimal subtask spanning trees (*MSST*s), where each *MSST* represents a minimal possible combination of available elements (nodes and links) that guarantees the success to execute this specific subtask (i.e., failure of any element in *MSST* leads to the subtask failure). By this definition of *MSST*, we can see that each *MSST* contains exactly one set of data resources without any duplications, because any duplication could be reduced to another smaller SST. Therefore, for any *MSST*, the data resources and precedent subtasks that provide certain input for the subtask are also determined. One can also obtain the working times of different elements by (12)-(15).

Some elements inside one *MSST* can still belong to several paths if they are involved in different communications tasks, such as data transmission or data resource access. Note that all elements in the execution stage are hot-standby although some elements/subtasks may be waiting for the output of some other subtasks. So during the waiting period, those elements are also possible to fail. Thus, we suppose that an MSST completes the entire service if all of its elements do not fail during the maximal time allowed to complete all subtasks in executing which they are involved. Therefore, when calculating the element reliability in a given MSST, one has to use the corresponding record with maximal time.

Assume there are a total of K elements in an *MSST*, and $element_i$ (i=1,2,…,K) denotes the i:th element in the *MSST*. Accordingly, the communication time of the i:th element is denoted by $T_w(element_i)$ and $\lambda(element_i)$ represents its failure rate. The reliability of this single *MSST* can be simply expressed as

$$R_{MSST} = \prod_{i=1}^{K} \exp\{-\lambda(element_i) \cdot T_w(element_i)\} \qquad (16)$$

With this equation, the reliability of an *MSST* can be computed if the working times of all the elements are obtained. Hence, finding all the *MSST*s and determining the working time of their elements are the first step in deriving the execution reliability of a cloud service. To solve the graph traversal problem, several classical algorithms have been suggested, such as depth-first search, breadth-first search, etc. These algorithms can find all *MSST*s in an arbitrary graph. Here, we propose a depth-first search algorithm here, which is briefly described as follows:

Step 1. Given a program/subtask, say S_m, start from a node that contains this program, to search the required data resources and precedent subtasks/programs along the possible links, and record elements that compose the searching route and their communication times.

Step 2. Until all the required data resources and precedent subtasks/programs are reached, an *MSST* is found, and record this *MSST*.

Step 3. Then other routes are tried to search other *MSST*s until all the *MSST*s are searched.

Step 4. Change to another node that also contains the program S_m. Repeat the above three steps until all the nodes that have S_m are evaluated. Save all the *MSST*s found associated with S_m into the vector $MSST(S_m)$.

Step 5. Change to another program and repeat the above four steps until all the programs are explored. Then all the vectors of $MSST(S_m)$ (m=1,2,…M) are generated.

B. Minimal Execution Spanning Tree (*MEST*)

Similar to the *MSST*, a Minimal Execution Spanning Tree (*MEST*) represents a minimal possible combination of available elements (nodes and links) that guarantees the success to execute the entire service. Thus, at least one *MSST* of each $MEST(S_m)$ (m=1,2,…M) must be reliable, and then the subtask S_m (m=1,2,…M) can be connected to those remote resources and exchange data with them successfully through the network. If any set of the M subtasks are successful, then the execution is reliable for the cloud service to execute the required set of subtasks, so the *MEST* could be derived as the intersection of the above sets of *MSST*s as

$$MEST = \bigcap_{m=1}^{M} MSST(S_m) \tag{17}$$

In practice, all *MEST*s could be generated in the following steps:

Step 1: Select an *MSST* from each set of $MSST(S_m)$ where (m=1,2,…M).

*Step 2: M MSST*s are obtained and put them together to generate the *MEST*. For each common element when intersecting trees together, record the greater working time as the final working time of this element in the *MEST*.

Step 3: Repeat Step 1-2 until all combinations are tried to generate all *N MSST*s.

Similar to (16), the reliability of a single *MEST* can be calculated by

$$R_{MEST} = \prod_{i \in MEST} \exp\{-\lambda(element_i) \cdot T_w(element_i)\} \tag{18}$$

C. Execution Reliability

Having the list of *N MEST*s and the corresponding task completion time, one can determine the reliability of cloud service at the execution stage, as follows.

$$R_{execute} = \Pr\left(\bigcup_{i=1}^{N} MEST_i\right) \tag{19}$$

which means any one *MEST* out of the total *N MEST*s being succeeded will make the cloud service successfully executed in the execution stage. Denote event E_j the successful operation of the $MEST_j$ while $\overline{E}_j$ the failure of the $MEST_j$. Using the Bayesian theorem on conditional probability, we can derive (19) to a summation of conditional probabilities

$$R_{execute} = \Pr\left(\bigcup_{i=1}^{N} MEST_i\right) = \sum_{j=1}^{N_i} \Pr(E_j) \cdot \Pr\left(\overline{E}_1, \overline{E}_2, \cdots, \overline{E}_{j-1} \middle| E_j\right) \tag{20}$$

The probability $\Pr(E_j)$ can be directly obtained from (18) as R_{MEST_j} and the probability $\Pr\left(\overline{E}_1, \overline{E}_2, \cdots, \overline{E}_{j-1} \middle| E_j\right)$ can be computed by the following two-step algorithm.

Step 1 identifies the failures of all of the critical elements in a period of time during which they lead to the failures of any one *MEST* from previous *j*-1 *MEST*s, but do not affect $MEST_j$.

Step 2 generates all the possible combinations of the identified critical elements that lead to the event $\overline{E}_1, \overline{E}_2, \cdots, \overline{E}_{j-1} | E_j$ by a binary search, and computes the probabilities of those combinations. Their summation is $\Pr\{\overline{E}_1, \overline{E}_2, \cdots \overline{E}_{j-1} | E_j\}$.

When calculating the failure probabilities of *MEST*s' elements the maximal time from the corresponding records in a list for the given *MEST* should be used.

Finally, if a cloud service needs to be successfully completed, both request stage and execution stage should be reliable. After we derive the reliability for both stages, we can hereby get the cloud service reliability $R_{Service}$ as

$$R_{Service} = R_{request}\, R_{execute} \tag{21}$$

where $R_{request}$ can be derived from the reliability of request stage by (10), and $R_{execute}$ can be derived from the reliability of execute stage by (20).

4. Conclusion and Discussion

In this paper, reliability modeling and analysis of cloud service is conducted. We first elaborate various types of possible failures in a cloud service, based on which a holistic reliability model is developed. A new algorithm is proposed to evaluate cloud service reliability based on the developed model.

The developed cloud service reliability model and evaluation algorithm, however, is yet to be validated by simulation and real-life data. This issue shall be addressed in our future research.

Acknowledgement:

This work is supported by National Science Foundation (No. 0831609) of USA. This work is supported by National Natural Science Foundation of China (No. 50805018) and Key Project of Chinese Ministry of Education (No. 109138).

References:

[1] I. Foster, C. Kesselman. The Grid 2: Blueprint for a New Computing Infrastructure. Los Alios, Morgan-Kaufmann, 2003.

[2] C.S. Yeo, R. Buyya1, M.D. de Assunção, et al. Utility Computing on Global Grids. Technical Report, GRIDS-TR-2006-7, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, 2006.

[3] Y. Zhang, Y. Zhou. Transparent computing: A new paradigm for pervasive computing. Proceedings of the 3rd International Conference on Ubiquitous Intelligence and Computing (UIC-06), LNCS 4145, 1–11, 2006.

[4] Y.S. Dai, Y. Pan, X.K. Zou. A hierarchical modeling and analysis for grid service reliability. *IEEE Transactions on Computers*, 56(5), 681-691, 2007.

[5] http://aws.amazon.com/ec2/

[6] http://www.xen.org/

[7] http://www.googlecloud.com/

[8] http://www.ibm.com/ibm/cloud/

[9] http://www.microsoft.com/azure

[10] M.L. Shooman. Reliability of Computer Systems and Networks: Fault Tolerance, Analysis and Design. New York: John Wiley & Sons, Inc., 2002.

[11] M. Xie, Y.S. Dai, K.L. Poh. Computing System Reliability: Models and Analysis. New York: Kluwer Academic Publishers, 2004.

[12] L. Xing, Y.S. Dai, "A new decision diagram model for efficient analysis on multi-state systems", *IEEE Transactions on Dependable and Secure Computing,* Accepted for Publication, 2008, Publishers: IEEE Press.

[13] X. Zou, Y.S. Dai, Y. Pan, *Trust and Security in Collaborative Computing*, World Scientific, Hackensack, NJ, U.S.A., 2008, ISBN: 981-270-368-3.

[14] D. Abramson, R. Buyya, J. Giddy. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. Future Generation Computer Systems, 18(8), 1061-1074, 2002.

[15] Y.S. Dai, M. Xie, K.L. Poh. Reliability of grid service systems, Computers & Industrial Engineering, 50(1-2), 130-147, 2006.

[16] Y.S. Dai, M. Xie, K.L. Poh, "Reliability Analysis of Grid Computing Systems", *The 9th IEEE Pacific Rim Symposium on Dependable Computing* (*PRDC2002*)*,* IEEE Computer Press, 2002, pp. 97-103.

[17] M. Xie, Y.S. Dai, K.L. Poh, *Computing Systems Reliability: Models and Analysis*, (330 pages), Springer: New York, U.S.A., 2004. ISBN: 0-306-48496-X.

[18] B. Yang, M. Xie. A study of operational and testing reliability in software reliability analysis, Reliability Engineering & System Safety, 70(3), 323-329, 2000.