An Novel Approach to Evaluate the Reliability of Cloud Rendering System Using Probabilistic Model Checker PRISM : A Quantitative Computing Perspective

Haoyu Liu[1,3], Huahu Xu[1,3*]
[1]School of Computer Engineering and Science, Shanghai University, 200444 Shanghai, China, liuhaoyu76@163.com
[2]Computing Center, Shanghai University, 200444 Shanghai, China, gaohonghao@shu.edu.cn
Honghao Gao[1,2], Minjie Bian[3], Huaikou Miao[1,4]
[3]Shanghai Shang Da Hai Run Information System Co. Ltd, 200444 Shanghai, China, huahuxu@163.com
[4]Shanghai Key Laboratory of Computer Software Testing & Evaluating, 201112 Shanghai, China, hkmiao@shu.edu.cn

ABSTRACT

This paper proposes an approach to evaluate the reliability of cloud rendering system. After the requirement analysis, the rendering system was divided into three modules: preparing files, requesting resources, and rendering task execution. Each module may have an exception that will reduce reliability, and has the ability to recover it. To expose these details, the discrete-time Markov chain (DTMC) is improved to formalize the cloud rendering system. The model contains an abnormal state set representing exceptions and errors such as file corruption and failure to rendering subtasks. Then, a series of formal properties are defined to describe reliability in detail. The proposed method gives full consideration to the processes of rendering tasks. Finally, the properties are verified by performing PRISM in a quantitative way. The experiment shows that our method is effective to evaluate the reliability of the cloud rendering system.

CCS Concepts

•**Software and its engineering** → Software verification; Software reliability;

Keywords

cloud rendering; DTMC model checking; PRISM; reliability

1. INTRODUCTION

Rendering is the relatively important step in the production of film and television cartoons. Cloud rendering [20,21,23] is a popular method used by filmmakers to perform rendering tasks to reduce time and save money. To ensure quality of service (QoS) [24,25], the service provider usually proavides a service level agreement to clarify the QoS parameters and responsibilities to be carried out. There are various definitions of QoS attributes [1,2,9]. Software reliability is the most commonly mentioned attribute. However, it is not perceivable by the client. Service providers need to test their system and verify reliability by themselves.

There are two ways to evaluate software reliability: the black-box method and the white-box method [23]. The black-box method uses a mathematical statistics method but does not consider the system internal structure and operational processes. It makes the reliability definition general, and the evaluation is not detailed. The cloud rendering system is complex, and it is not sufficient to define and verify reliability from only a holistic perspective. However, the white-box approach is based on the path and state of the system model and uses model checking for a quantitative analysis. It takes full account of the details.

In this paper, we use a white-box approach, and improved a model to evaluate the reliability of cloud rendering systems. Compared to other models, this model is useful for presenting more detailed reliability attributes. Based on the approach, the reliability checking results will become more specific and more useful for analyzing the reliability of cloud rendering systems. Our work is divided into the following points. First, an extended discrete-time Markov chain (DTMC) model is proposed. In the proposed model, the states are divided into two categories: normal and abnormal. The set of abnormal states contains some error states describing the cause of failure. Then, nine reliability properties are defined from three aspects, including success rate, exception rate and recovery rate. Where the success rate aspect analyzes mainly the running results of the entire system, the other two are aimed at analyzing specific function modules. Finally, these properties are verified by using PRISM [6,7]. The experimental results are used to evaluate the reliability of the three aforementioned aspects. Experiments show that the proposed method can describe the reliability of a cloud rendering system in more detail and guarantee the validity of the reliability check.

The rest of this paper is structured as follows: Section 2 describes the related works. Section 3 proposes an extended DTMC to model the cloud rendering system. Section 4 defines a set of reliability properties and corresponding formulas in PRISM. Section 5 verifies the effectiveness of the proposed method by experiment. Section 6 summarizes the research contents of this paper and suggests a future research direction.

MobiQuitous 2017, November 7–10, 2017, Melbourne, VIC, Australia

ACM ISBN 978-1-4503-5368-7/17/11$15.00
https://doi.org/10.1145/3144457.3144460

2. RELATED WORKS

The rendering task is computing and data intensive. Traditional stand-alone rendering methods are inefficient and unreliable. Some remote rendering systems [3,4,10] solve the rendering problem, but those systems are not useful for large-scale rendering tasks. One proposal for a large rendering system was based on Hadoop [14]. It stores every frame of the scene on HDFS and uses the MapReduce programming model to complete job scheduling, parallelization, etc. For ultra-large-scale cloud rendering, Weini Zhou et al. [15] proposed a five-level software architecture, including the infrastructure, platform service, and application service layers, service management, and an access interface. The architecture of the cloud rendering system is complex, and it is difficult to define and verify its reliability.

Software reliability is the most commonly mentioned factor in QoS, and there are two ways to evaluate it: the black-box method and the white-box method. For the black-box method, Ali Hamlili [16] proposed an efficient reliability evaluation by several black-box models of reliability growth. Youngjun Choe et al. [17] developed a new approach, called stochastic importance sampling (SIS), which efficiently uses stochastic simulations with unknown output distribution. However, the black-box approach is not suitable for evaluating complex systems, because it disregards system details. In contrast, the white-box method evaluates the reliability of the individual components according to the system details. Nagarajan Padmavathy et al. [18] proposed a Monte Carlo Simulation based evaluation of mobile ad hoc network reliability which considers different mobility models along with the effect of different scenario metrics and different values of tuning parameter. Young Myoung Ko et al. [19] propose a new methodology that allows us to overcome difficulties in analyzing large-scale system dynamics, and devise analytical methods for finding the multivariate distribution of the dynamically changing system condition. Models in these approaches are all based on the internal structure of the software. However, the definition of the reliability property is not detailed. In this study, we proposed a property model by using the white-box approach to obtain more reliability properties.

Probability model checking [28] is used to validate a class of system models with Markovian properties. That stochastic process was proposed by the Russian mathematician Andrey Markov in 1907, mainly in the following categories: the DTMC, the continuous-time Markov chain (CTMC) [27], and the Markov decision process (MDP) [5]. The properties of the model are described by the property specification, which mainly includes probabilistic computation tree logic (PCTL) [11], linear temporal logic (LTL) [12], and continuous stochastic logic (CSL) [13]. There are many tools to verify the properties. The Markov reward model checker [8] is a command line probabilistic model test tool. It uses coefficient-matrix-based data structures and algorithms and supports accurate on-the-fly steady-state checks. It is more efficient for a small-scale stochastic system model. PRISM was developed in Java / C++. It is a graphical stochastic model checking tool that can be run on a variety of mainstream operating systems, such as Windows, Linux, MacOS, and so on. PRISM can verify DTMC, CTMC, MDP, and other models, and supports PCTL, CSL, and other property specification languages. Therefore, in this study we used PRISM to check the reliability properties of the rendering system in the experiment.

3. PROBABILITY MODEL

The cloud rendering system is based on cloud computing technology, which has the following characteristics: a need for large storage nodes to save rendered files, a need for powerful CPUs and GPUs to accommodate the large number of rendering calculations, and a need for excellent task scheduling to ensure that the rendering task is executed in a timely manner. With these characteristics, the operation of the cloud rendering system is divided into several steps: checking rendered files, requesting cloud computing resources, assigning rendering tasks, performing subtasks, merging results, etc. Fig. 1 details the implementation process.

At the beginning of the rendering task, files are checked. If some files are damaged, the system calls a repair program to fix them. For the request resources step, it is important to note that cloud computing resources do include computing nodes and storage nodes, however those resources are limited, and until they are sufficient, the current task must wait. Another problem is that the rendering subtasks are not always performed successfully. A variety of hardware or software errors is likely to lead to failure. So at the end of the task, the system must check whether all subtasks were executed successfully. If some of the executions failed, the system must assign failed subtasks to re-execute. When all rendering subtasks have been executed successfully, the system combines the results.

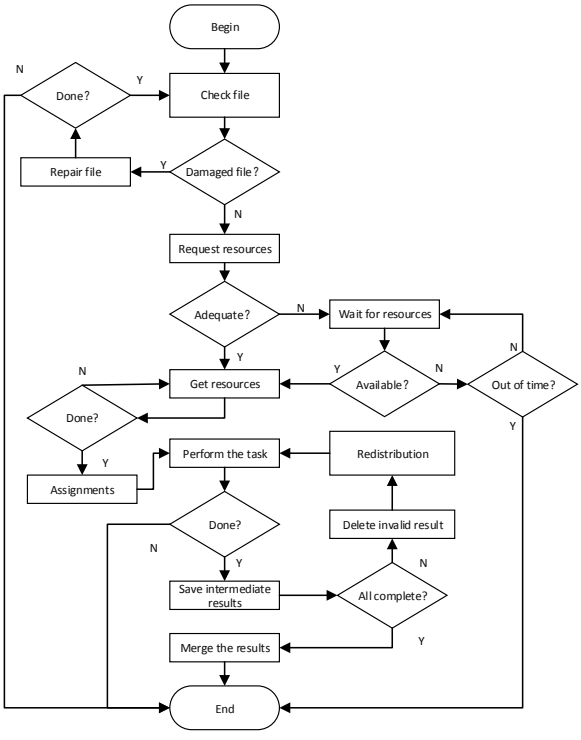

Figure 1: Implementation process

3.1 Extended DTMC

To help verify reliability properties, the DTMC model is extended by dividing the state set into a normal set and an abnormal set. The execution step corresponds to the state. The extended DTMC is defined as

$$M = (S_{normal}, S_{abnormal}, s_{init}, P, L) \quad (1)$$

where S_{normal} is a set of finite nonempty states that contains all normal states, the initial state, and the success state. $S_{abnormal}$ is also a set of finite nonempty states, but contains all the exception states and error states. s_{init} is the initial state of the system and $s_{init} \in S_{normal}$. $L : S \to 2^{AP}$ is a label function, which is the mapping of the atomic proposition set. AP is the atomic proposition set, which marks implementation steps and the errors system. $P : S \times S \to [0,1]$ is a state transition function, expressed as a matrix, which is used to indicate the probability that a state will transfer to another state.

3.2 Rules of Model Transformation

- Sequential execution means that state transitions have no branches, so the probability of the transition is 1. The conversion rules are shown in Fig. 2.

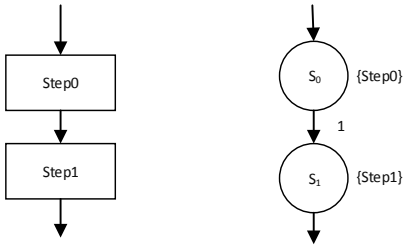

Figure 2: Sequential execution

- In the conditional judgments, there are two branches to perform different operations. Fig. 3 shows that two paths can be used to represent the conditional branches, and the sum of the transition probabilities is 1.

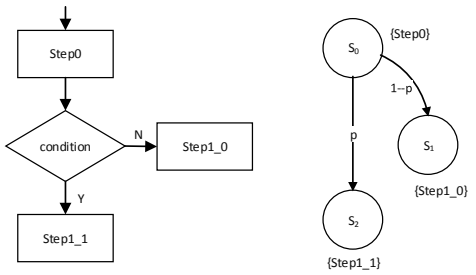

Figure 3: Condition branch

- If a branch in the conditional judgment returns to the previous step, the branch forms a loop. In the model, the loop can be described by a self loop. The sum of the probabilities is also 1. Fig. 4 shows the relationship.

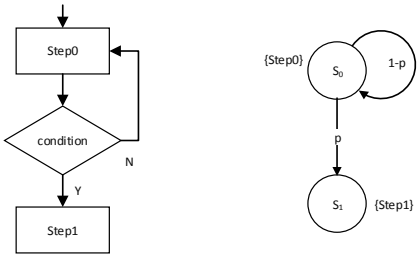

Figure 4: Cycle operation

3.3 Modeling the Cloud Rendering System

After the analysis of the running process of the cloud rendering system, the system is divided into three function modules: preparing files, requesting cloud computing resources, and performing rendering tasks. They are similar to the running process; that is, the system will take measures to return to a normal state when it enters an abnormal state. Therefore, we can use the modeling rules as a basis for building the model for each function module.

Before the modeling, some states are defined as follows: s_{succ} is a success state, s_{fail} is a failure state, and other states are represented by $s(a,b)$, where a is the serial number of the function module and b is the serial number of the execution step or error state. Table 1 shows detailed definitions of states in the cloud rendering system.

Table 1: State Definitions

No.	Module	State Name	$s(a,b)$
1	Prepare File	Check File (normal)	$s(1,1)$
		Repair File (abnormal)	$s(1,2)$
		Invalid File (error)	$s(1,3)$
2	Request Cloud Computing Resources	Request Resources (normal)	$s(2,1)$
		Wait for Resources (abnormal)	$s(2,2)$
		Get Resources (normal)	$s(2,3)$
		Lack of Resources (error)	$s(2,4)$
3	Perform the rendering task	Assign Rendering Subtasks (normal)	$s(3,1)$
		Perform Rendering Subtasks (normal)	$s(3,2)$
		Save Intermediate Results (normal)	$s(3,3)$
		Delete Invalid Results (abnormal)	$s(3,4)$
		Reassign Subtasks (abnormal)	$s(3,5)$
		Merge Intermediate Results (normal)	$s(3,6)$
		Task Execution Failed (error)	$s(3,7)$

The following is an explanation of the operation process and the corresponding model of each functional module.

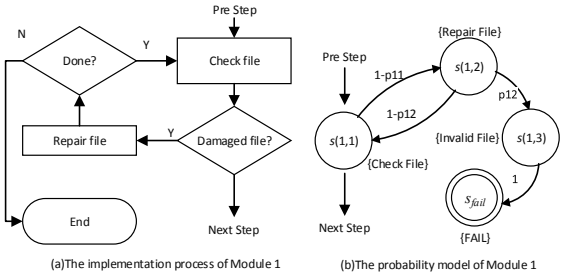

Figure 5: Model build process for function module 1

Function module 1 is mainly to verify file integrity and repair damaged files. In Fig. 5, there are two execution steps in the process, which correspond to $s(1,1)$ and $s(1,2)$. However, an exception may occur in the file checking step. So the system must perform a fix operation to return to the normal state. If the operation fails, the cloud rendering system cannot continue to run and enters the failure state s_{fail}. Before jumping to s_{fail}, an error state $s(1,3)$ is added to explain that the failure was caused by file corruption.

Function module 2 mainly requests cloud computing resources. Fig. 6 shows the modeling process. Function module 2 consists of three steps: request, wait, and get resources, which correspond to $s(2,1)$, $s(2,2)$, $s(2,3)$. The system will go into a waiting state when some resources are occupied by other rendering tasks and there are not enough resources to be allocated. If the waiting time exceeds the maximum limit, the system reports an error and the operation fails. So $s(2,4)$ is added before the failure state s_{fail} to explain that the cloud computing resource is insufficient.

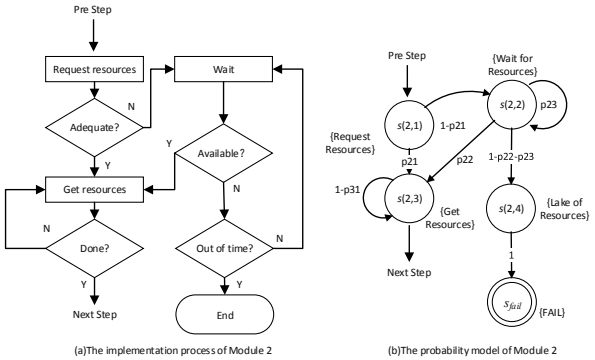

Figure 6: Model build process for function module 2

Function module 3 is the implementation of rendering. Fig. 7 shows the modeling process. Function module 3 includes the following steps: assign rendering tasks, execute subtasks, save intermediate results, merge rendering results, etc. These steps correspond to $s(3,1)$, $s(3,2)$, $s(3,3)$, $s(3,4)$, $s(3,5)$, and $s(3,6)$. However, the computing nodes are unable to perform the assigned rendering subtasks successfully every time, because of some compatibility issues. Even if the subtask is executed without error, there is no guarantee that the results are correct. So the system will attempt to reassign the failed rendering subtasks to the computing nodes. If the number of attempts exceeds the maximum allowed, the system enters state $s(3,6)$ to merge only correct intermediate results. That is, only part of the intermediate results are merged. This is also considered a failure, so state $s(3,6)$ still has some probability of migrating to the error state $s(3,7)$.

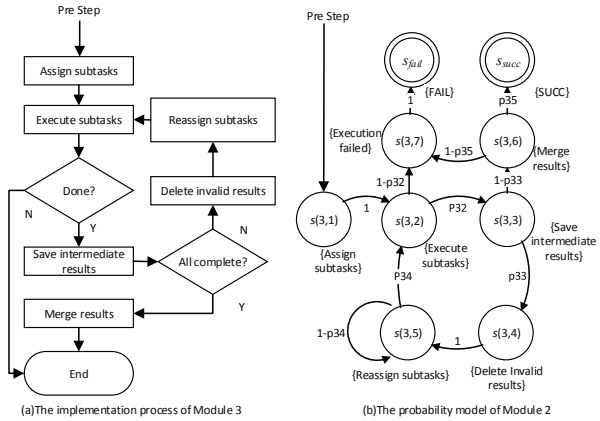

Figure 7: Model build process for function module 3

According to the above model of each function module, the combined model is shown in Fig. 8, which represents the entire cloud rendering system.

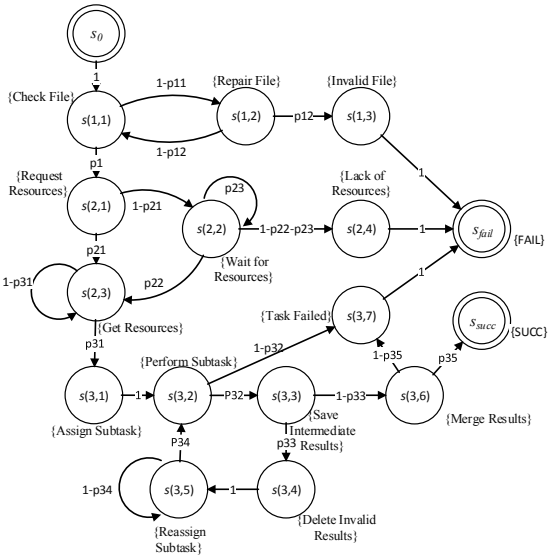

Figure 8: Probability model of cloud rendering system

Then the combined model is formalized by using the extended DTMC model proposed before. The definitions of its parameters are shown as

$S_{normal} = \{s_0, s(1,1), s(2,1), s(2,3), s(3,1), s(3,2), s(3,3), s(3,6), s_{succ}\}$

$S_{abnormal} = \{s(1,2), s(1,3), s(2,2), s(2,4), s(3,4), s(3,5), s(3,7), s_{fail}\}$

$s_{init} = s_0$

$L(s(1,1)) =$ { Check File }

$L(s(1,2)) =$ { Repair File }

$L(s(1,3)) =$ { Invalid File }

$L(s(2,1)) =$ { Request Resources }

$L(s(2,2)) =$ { Wait for Resources }

$L(s(2,3)) =$ { Get Resources }

$L(s(2,4)) =$ { Lack of Resources }

$L(s(3,1)) =$ { Assign Rendering Subtasks }

$L(s(3,2)) =$ { Perform Rendering Subtasks }

$L(s(3,3)) =$ { Save Intermediate Results }

$L(s(3,4)) =$ { Delete Invalid Results }

$L(s(3,5)) =$ { Reassign Rendering Subtasks }

$L(s(3,6)) =$ { Merge Intermediate Results }

$L(s(3,7)) =$ { Task Execution Failed }

$$P=\begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & p11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\text{-}p11 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1\text{-}p21 & p21 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & p23 & p22 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\text{-}p22\text{-}p23 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1\text{-}p31 & p31 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & p32 & 0 & 0 & 0 & 0 & 0 & 0 & 1\text{-}p32 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p33 & 0 & 1\text{-}p33 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & p34 & 0 & 0 & 1\text{-}p34 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p35 & 0 & 0 & 1\text{-}p35 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1\text{-}p12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p12 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}$$

4. RELIABILITY PROPERTIES

4.1 Reachability

In our paper, the reliability description is based on reachability [30]. Reachability refers to the ease with which a node arrives at another node in the model. It can be described by PCTL formula [26]. PCTL is a probabilistic branch temporal logic that can be used to qualitatively describe the property of the Kripke structure model. More convenient, the PCTL formula can be checked in the supporting tool PRISM.

Next, we will present a series of reliability properties and convert them into PCTL formulas into PRISM. In this way, the PRISM tool is employed to check the proposed properties in the experiment.

4.2 Success Rate

If the cloud rendering system runs successfully, it means all subtasks have been completed and produced valid results. However, two cases can cause system failure. One case is that a system error causes the program to terminate early. Another is that only a portion of the subtasks are completed in the end, which is called part-complete. Therefore, according to the running result, the properties that follow are proposed.

Property 1: The probability of success is greater than 95%.

Property 2: The probability of failure is less than 1%.

Property 3: The probability of part-complete is less than 1%.

In PRISM, properties 1-3 are represented by (2)-(4).

$$P = ?\ [\ F <= step\ (exception <= upper\ \&\ all\_complete = true)\ ] \tag{2}$$

$$P = ?\ [\ F <= step\ (exception <= upper\ \&\ s = 15\ \&\ part\_complete = false)\ ] \tag{3}$$

$$P = ?\ [\ F <= step\ (exception <= upper\ \&\ part\_complete = true)\ ] \tag{4}$$

where *step* is the number of steps that the state of the system migrated, *exception* indicates the total number of times the system entered an abnormal state, *upper* is the maximum number of times the system was allowed to enter the abnormal state, *all_complete* indicates whether the system completed all rendering subtasks, and *part_complete* indicates whether the system only partially completed subtasks.

4.3 Exception Rate

From the perspective of the exception rate, three cases lead the system into the abnormal state:

1) The file may be corrupted during the transfer and storage process. If the file is damaged, the system goes into an abnormal state and tries to repair the file.

2) When many cloud computing resources are occupied by other tasks, the current task is suspended and waits for a long time. This means that the system has entered an abnormal state.

3) The compute node cannot guarantee that all subtasks are executed successfully because the performance of the node is different. If some subtasks fail, the system goes into an abnormal state and tries to reassign those subtasks.

Through the analysis of the above problems, the following three reliability properties are proposed.

Property 4: The probability that the file is damaged is less than 1%.

Property 5: The probability that the resources are insufficient is less than 1%.

Property 6: The probability that the subtask fails to execute is less than 1%.

In PRISM, properties 4-6 are represented by (5)-(7).

$$P = ?\ [\ F <= step\ (repair\_file > 0\ \&\ exception > 0)\ ] \tag{5}$$

$$P = ?\ [\ F <= step\ (wait\_resources > 0\ \&\ exception > 0)\ ] \tag{6}$$

$$P = ?\ [\ F <= step\ (rerendering > 0\ \&\ exception > 0)\ ] \tag{7}$$

where *step* and *exception* are defined as they were previously, *repair_file* is the number of times the system has repaired the file, *wait_resources* is the number of times the system waited for resources, and *rerendering* is the number of times the system re-executed the failed subtasks.

4.4 Recovery Rate

The recovery rate is the ability of the system to transfer from an abnormal state to a normal state; this represents the reliability.

The properties that follow describe the recovery ability for the cloud rendering system.

Property 7: When the rendered file is corrupted, the probability that the system fixes the file is greater than 95%.

Property 8: When a task is waiting for computing resources for a long time, the probability that the system timely allocates resources to the specific task is greater than 95%.

Property 9: When the node fails to execute the subtask, the probability that the system re-executes failed subtasks and eventually succeeds is greater than 95%.

In PRISM, properties 7-8 are represented by (8)-(10).

$$\frac{P=?\ [\ F<=step\ (file\_ok=true\ \&\ repair\_file>0)\ ]}{P=?\ [\ F\ (repair\_file>0\ \&\ exception>0)\ ]} \tag{8}$$

$$\frac{P=?\ [\ F<=step\ (wait\_resources>0\ \&\ resources\_ok=true)\ ]}{P=?\ [\ F\ (wait\_resources>0\ \&\ exception>0)\ ]} \tag{9}$$

$$\frac{P=?\ [\ F<=step\ (rerendering>0\ \&\ all\_complete=true)\ ]}{P=?\ [\ F(rerendering>0\ \&\ exception>0)\ ]} \tag{10}$$

where *file_ok* indicates whether the file is damaged or ready to use, *resources_ok* indicates whether the current task has enough cloud computing resources to perform rendering tasks, and other parameters are defined as they were previously.

5. EXPERIMENT AND ANALYSIS

The proposed model is versatile that most cloud rendering systems can use it to verify reliability. For real cloud rendering system, it should be necessary to determine the state in the program and record the number of state branch jumps. In the experiment, the cloud rendering system consisted of 20 computers, and the probability parameters were obtained from system logs. They are formalized in PRISM as shown in Fig. 9.

```
1   const double p11=0.9858;
2   const double p21=0.9892;
3   const double p23=0.9537;
4   const double p22=0.0173;
5   const double p31=0.9776;
6   const double p32=0.9858;
7   const double p12=0.0189;
8   const double p33=0.0143;
9   const double p34=0.9889;
10  const double p35=0.9829;
```

Figure 9: Probability parameters in PRISM

Fig. 10 shows the results of the checking on property 1. The probability that the system would run successfully without any exceptions was less than 95%. However, the probability was greater than 95% when only one exception was allowed to occur and was close to 97% when more than two exceptions were allowed to occur. Therefore, the higher the allowed number of exceptions, the higher the probability of success. That probability is because the system had the ability to recover from exceptions. So the system satisfied property 1, meaning the probability of success was greater than 95%.

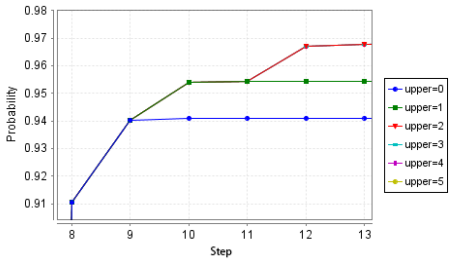

Figure 10: Results of checking on property 1

Fig. 11 shows the results of the checking on property 2. There was a small probability that failure would occur within six steps. After that, the probability was markedly higher. On the other hand, as the allowed number of exceptions increased, the failure rate was raised because the recovery operations may have failed. For the current system, if the allowed number of exceptions exceeded three, the probability of failure would be close to 1.5%. Therefore, the system did not meet property 2; that is, the failure rate was not less than 1%.

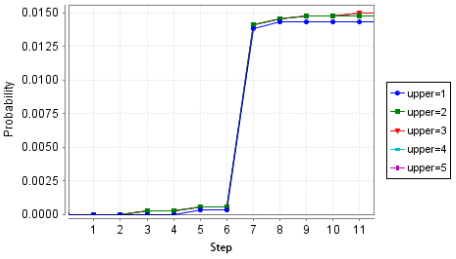

Figure 11: Results of checking on property 2

Fig. 12 shows the results of the checking on property 3. As can be seen from the figure, the probability of part-completed increased with the migrations of the state. If the system allowed only one exception to occur, the probability of part-complete exceeded 1.6%. When the allowed number of exceptions was greater than two, the probability still rose slightly because the cloud rendering system had more opportunities to re-execute the failed rendering subtasks and was more prone to part-complete. In any case, the system did not meet property 3. The probability of part-complete was greater than 1%.

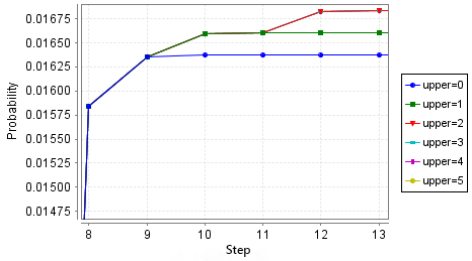

Figure 12: Results of checking on property 3

Fig. 13 shows the results of the checking on properties 4-6. Obviously, the file corruption exception appeared earlier, and the rendering exception occurred later. That sequence was related to the order in which they were executed. The probability of an exception caused by lack of cloud computing resources was low at less than 0.25%. However, the probability of a rendering exception and a file exception was greater at more than 1.25%. So the system satisfied property 5, but did not meet properties 4 and 6, which means that the probabilities of a file exception and a rendering exception were not less than 1%.

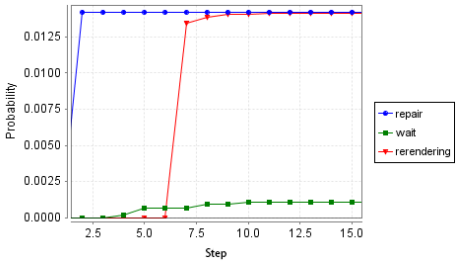

Figure 13: Results of checking on properties 4–6

Fig. 14 shows the results of the checking on properties 7-9. The file exception could be fixed at the earliest, and the rendering exception was fixed at the latest. That sequence was also consistent with the order of operations. However, the probability of successful recovery for the file exception was greater than 97% and for the other two exceptions was also close to 97%. So the cloud rendering system satisfied properties 7-9. That is, the success rate of the exception recovery was greater than 95%.

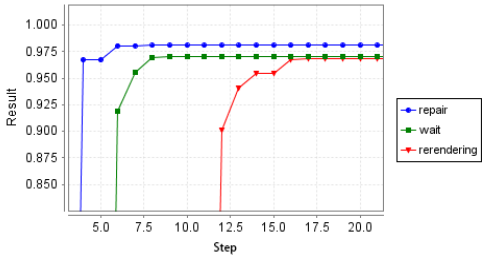

Figure 14: Results of checking on properties 7–9

In conclusion, the experimental cloud rendering system satisfied five properties. The evaluation and improvements follows.

1) From the perspective of the results, the success rate was 94% when the system performed rendering tasks without any exceptions occurring. However, as the allowed number of exceptions increased, the success rate became higher. The probability of success was close to 97% when the maximum number of exceptions was greater than two. On the other hand, the more exceptions that occurred, the worse the stability of the system. So to ensure the success rate and stability of the system, it was necessary to adjust the maximum number of exceptions to two.

2) From the perspective of the exception rate, the probability of a file corruption exception was relatively high. That probability shows that there may have been loopholes in the file transfer or file access program module. It is also possible that the quality of the storage node was not up to standard. For file exceptions, this lack of quality required upgrading the configuration of storage nodes and improving the file transfer and access programs. In addition, the probability of a rendering exception was as high as that of a file exception. That probability may have been due to an incompatibility of the rendering engine or the poor calculating ability of the CPU. It is effective to use a more compatible rendering engine or to replace inefficient computing nodes with higher performance computers. The probability of an exception caused by lack of cloud computing resources was low. That probability shows that the system had sufficient computing nodes and storage nodes to meet the needs of tasks.

3) From the perspective of the recovery rate, when a file was damaged, the file would be fixed by the system with a high probability. That repair indicates that the system functioned well in file backup and repair. When the current task waited for cloud computing resources, the probability that the system would assign the resources timely was also high, because the system was excellent at scheduling resources. Similarly, the probability of recovery of rendering exceptions was high due to the efficient strategy of task scheduling. In general, the system had a strong ability to return from the abnormal state to the normal state. In this experimental environment, this aspect did not need to be improved.

6. CONCLUSION AND FUTURE WORK

This paper proposes a method to evaluate the reliability of a cloud rendering system by using a probability model. The extended DTMC is proposed, and the probability model of a cloud rendering system was constructed. Then a series of reliability properties are defined from three aspects. In the experiment, we built a cloud rendering system with 20 computers and used PRISM to check the reliability properties. Finally, the reliability of the cloud rendering system is evaluated according to the experimental results. These results show that our method is effective to evaluate the reliability of the cloud rendering system.

Future work entails obtaining more accurate experimental results by increasing the number of computing nodes and rendering tasks. However, it is difficult to evaluate and improve the reliability of the cloud rendering system by artificial means. How to calculate reliability properties by the system itself, and how to use it to implement self-evaluation and self-optimization will be studied in the future.

ACKNOWLEDGMENT

This paper is supported by National Natural Science Foundation of China under Grant No. 61502294, 61572306, Natural Science Foundation of Shanghai under Grant No.15ZR1415200, CERNET Innovation Project under Grant No.NGII20160210, NGII20160614, NGII20160325, The Special Development

Foundation of Key Project of Shanghai Zhangjiang National Innovation Demonstration Zone under Grant No. 201411-ZB-B204-012, and The Development Foundation for Cultural and Creative Industries of Shanghai under Grant No. 201610162.

REFERENCES

[1] Y. Chen, et al. 2016. Multi-Objective Service Composition with QoS Dependencies. IEEE Transactions on Cloud Computing PP.99, 1-1.

[2] M. Rady. 2013. Formal Definition of Service Availability in Cloud Computing Using OWL. Computer Aided Systems Theory - EUROCAST 2013, Springer Berlin Heidelberg, 189-194.

[3] S. Shi, C. H. Hsu. 2015. A survey of interactive remote rendering systems. Acm Computing Surveys, 47(4), 57.

[4] P. Quax, J. Liesenborgs, A. Barzan, M. Croonen, W. Lamotte, B. Vankeirsbilck, et al. 2016. Remote rendering solutions using web technologies. Multimedia Tools & Applications, 75(8), 1-28.

[5] J. Kim, et al. 2014. MDP based dynamic base station management for power conservation in self-organizing networks. Wireless Communications and NETWORKING Conference IEEE, 2384-2389.

[6] A. Hinton, et al. 2006. PRISM: A Tool for Automatic Verification of Probabilistic Systems. International Conference on Tools and Algorithms for the Construction and Analysis of Systems Springer Berlin Heidelberg, 441-444.

[7] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. Computer Aided Verification - International Conference, CAV 2011, Snowbird, Ut, Usa, 14-20. Proceedings DBLP, 585-591.

[8] L. Cloth, J. P. Katoen, M. Khattri, and R. Pulungan. 2017. Model checking markov reward models with impulse rewards, 722-731.

[9] B. Alexander, K. Marios, and H. Thoma. 2011. Service Quality in Software-as-a-Service: Developing the SaaS-Qual Measure and Examining Its Role in Usage Continuance. Journal of Management Information Systems, 85-126.

[10] P. Quax, J. Liesenborgs, A. Barzan, M. Croonen, W. Lamotte, B. Vankeirsbilck, et al. 2016. Remote rendering solutions using web technologies. Multimedia Tools & Applications, 75(8), 1-28.

[11] H. Fecher, M. Huth, N. Piterman, and D. Wagner. 2010. Pctl model checking of markov chains: truth and falsity as winning strategies in games. Performance Evaluation, 67(9), 858-872.

[12] M. Guo, D. V. Dimarogonas. 2015. Multi-agent plan reconfiguration under local ltl specifications. International Journal of Robotics Research, 34(2), 218-235.

[13] L. Zhang, et al. 2011. Automata-based CSL model checking. International Conference on Automata, Languages and Programming Springer-Verlag, 271-282.

[14] W. Liu, B. Gong, and Y. Hu. 2011. A large-scale rendering system based on hadoop. International Conference on Pervasive Computing and Applications IEEE, 2011, 470-475.

[15] W. Zhou, et al. 2012. A New Software Architecture for Ultra-large-scale Rendering Cloud. International Symposium on Distributed Computing and Applications To Business, Engineering & Science IEEE, 2012, 196-199.

[16] A. Hamlili. 2006. Reliability Evaluation and Prediction of Improvable Information and Communication etworks. Information and Communication Technologies. Ictta '06. IEEE, 3587-3592.

[17] Y. Choe, E. Byon, and N. Chen. 2015. Importance Sampling for Reliability Evaluation With Stochastic Simulation Models. Technometrics, 57(3), 351-361.

[18] N. Padmavathy, S. K. Chaturvedi. 2015. Reliability Evaluation of Mobile Ad Hoc Network: With and without Mobility Considerations. Procedia Computer Science, 46,1126-1139.

[19] Y. M. Ko, E. Byon. 2015. Reliability evaluation of large-scale systems with identical units. IEEE Transactions on Reliability, 64(1), 420-434.

[20] K. Mukhina, A. Bezgodov. 2015. The Method for Real-time Cloud Rendering. Procedia Computer Science, 66, 697-704.

[21] L. Zhu, A. Yue, and C. Zhou. 2015. A fast rendering method for 3d city building point cloud models. Jisuanji Fuzhu Sheji Yu Tuxingxue Xuebao/Journal of Computer-Aided Design and Computer Graphics, 27(8), 1443-1451.

[22] K. K. Mohan, A. K. Verma, A. Srividya, et al. 2010. Integration of Black-Box and White-Box Modeling Approaches for Software Reliability Estimation. International Journal of Reliability Quality & Safety Engineering, 17(03), 261-273.

[23] J. R. Annette, W. A. Banu, and P. S. Chandran. 2015. Rendering-as-a-Service: Taxonomy and Comparison. Procedia Computer Science, 50, 276-281.

[24] S. Singh, I. Chana. 2015. QoS-Aware Autonomic Resource Management in Cloud Computing: A Systematic Review. Acm Computing Surveys, 48(3), 1-46.

[25] B. G. Batista, C. H. G. Ferreira, D. C. M. Segura, et al. 2016. A QoS-driven approach for cloud computing addressing attributes of performance and security. Future Generation Computer Systems, 68, 260-274.

[26] S. Akshay, T. Antonopoulos, J. Ouaknine, et al. 2015. Reachability problems for Markov chains. Information Processing Letters, 115(2), 155-158.

[27] M. Iannelli, A. Pugliese. 2014. Continuous-time Markov chains. An Introduction to Mathematical Population Dynamics. Springer International Publishing, 135-164.

[28] J. P. Katoen. 2013. Model checking meets probability: a gentle introduction. Engineering Dependable Software Systems.

[29] C. Daws. 2005. Symbolic and Parametric Model Checking of Discrete-Time Markov Chains. Theoretical Aspects of Computing - ICTAC 2004. Springer Berlin Heidelberg, 280-294.

[30] E. M. Hahn, H. Hermanns, and L. Zhang. 2011. Probabilistic reachability for parametric markov models. International Journal on Software Tools for Technology Transfer, 13(1), 3-19.