On Cloud Service Reliability Enhancement with Optimal Resource Usage

A. Zhou, S. Wang, *Member, IEEE*, Z. Zheng, *Member, IEEE*, C. Hsu, *Senior Member, IEEE*, Michael R. Lyu, *Fellow, IEEE*, and F. Yang, *Member, IEEE*

Abstract—An increasing number of companies are beginning to deploy services/applications in the cloud computing environment. Enhancing the reliability of cloud service has become a critical and challenging research problem. In the cloud computing environment, all resources are commercialized. Therefore, a reliability enhancement approach should not consume too much resource. However, existing approaches cannot achieve the optimal effect because of checkpoint image-sharing neglect, and checkpoint image inaccessibility caused by node crashing. To address this problem, we propose a cloud service reliability enhancement approach for minimizing network and storage resource usage in a cloud data center. In our proposed approach, the identical parts of all virtual machines that provide the same service are checkpointed once as the service checkpoint image, which can be shared by those virtual machines to reduce the storage resource consumption. Then, the remaining checkpoint images only save the modified page. To persistently store the checkpoint image, the checkpoint image storage problem is modeled as an optimization problem. Finally, we present an efficient heuristic algorithm to solve the problem. The algorithm exploits the data center network architecture characteristics and the node failure predicator to minimize network resource usage. To verify the effectiveness of the proposed approach, we extend the renowned cloud simulator Cloudsim and conduct experiments on it. Experimental results based on the extended Cloudsim show that the proposed approach not only guarantees cloud service reliability, but also consumes fewer network and storage resources than other approaches.

Index Terms—Cloud service, reliability, optimization, cloud data center, network resource, storage resource

1 Introduction

Recently, cloud computing has emerged as a new paradigm for offering computing as services via the Internet [1], [2]. Many companies are beginning to deliver cloud application services/applications to lower the cost of maintaining their own computing infrastructure. Unfortunately, cloud data center downtime has badly affected the public's expectation of cloud computing. With tens of thousands of host servers in a cloud data center, it is difficult to ensure that all host servers always work well. Therefore, a statistically rare failure event may become common in a cloud data center [3]. Moreover, cloud computing adopts a multi-tenancy model in which all cloud service providers share with each other the same physical infrastructure. Unlike other computing models, a downtime incident in a cloud data center may cause serious damage to many cloud service providers. Therefore, the issue of how to enhance the cloud service reliability when the host servers fail has become a critical problem [4], [5].

In recent years, many fault tolerance approaches have been proposed to enhance cloud service reliability [6], [7]. Most of these approaches are based on exploitation of redundancy. Replication and checkpointing are two widely used basic mechanisms. In replication mechanism [8], [9], [10] the same task is synchronously or asynchronously processed on several virtual machines (VM). It only ensures that at least one replica is able to complete the task on time. However, the replication mechanism is more suitable for critical or real-time services [11], [12], [13] because of its large implementation cost. The checkpointing mechanism periodically saves the execution state of a running task as a checkpoint image file [14], [15]. When the server crashes, it can resume the task on a different server based on the latest saved checkpoint image. The task does not need to be restarted from the beginning but only from the latest saved state. Therefore, it can reduce lost time caused by the failure and improve the cloud service reliability.

State-of-the-art solutions [16], [17] have been proposed by extending the basic checkpointing mechanisms. Because all resources in cloud computing are commercialized, the goal of these solutions is to reduce resource consumption based on data center characteristics while enhancing cloud service reliability.

In checkpointing, a significant amount of data must be periodically transferred to persistently store checkpoint images. A checkpoint image can be a gigabyte size, and it contains all information to restart the service in another host server. In traditional approaches, because data center network resource is limited, checkpoint traffic may congest the network and affect QoS of other cloud services. To address this problem, smart checkpointing approaches have been proposed [14], [15], [16]. They provide the notification that a small percentage of data have changed

- *A. Zhou, S. Wang, F. Yang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. E-mail: {hellozhouao@gmail.com; sgwang@bupt.edu.cn; fcyang@bupt.edu.cn}.*
- *Z. Zheng and Michael R. Lyu are with Department of Computer Science & Engineering, The Chinese University of Hong Kong. E-mail: {zbzheng; lyug}@cse.cuhk.edu.hk*
- *C. Hsu is with the Computer Science and Information Engineering, Chung Hua University, Hsinch, Taiwan. E-mail: chh@chu.edu.tw*

since the last checkpointing time. The checkpoint images are classified into two subsidiary sets: system checkpoint image and delta checkpoint image. The checkpoint image containing the base system is called the system checkpoint image. The remaining checkpoint images, which contain only the modified data, are called the delta checkpoint images. Use of these subsidiary sets reduces the bandwidth required to transfer the checkpoint images to the central storage server.

While the checkpoint images are stored in the central storage server [15], [16], however, the checkpoint traffic must still be routed by the core switches. Core switches play an important role in the cloud data center by bridging the cloud data center and the outside world. Core switches, therefore, are the bottleneck of the cloud data center network. To prevent checkpoint traffic from congesting core switches, Limrungsi et al [17] proposed a distributed delta checkpointing approach. The VM images are not stored on the central storage servers but on neighboring host servers. Therefore, the images may only require to be transferred by the aggregation and edge switches.

Although the above approaches can reduce network resource usage, some limitations of them remain. For one, these approaches do not consider reducing storage resource consumption through system checkpoint image sharing. Because the base system, ram disk content, and code page are the same among the VMs, similarity exists among the system images generated from the VMs that provide the same service. Therefore, there is no need to save a system image copy for each VM. Second, when the checkpoint images are stored in neighboring host servers, host server crashes may result in the checkpoint image inaccessibility.

In this paper, we propose a reliability-aware distributed image-sharing checkpoint approach for minimizing network and storage resource usage called RADS-CKP (reliability-aware, distributed storage checkpointing). First, the checkpoint images are classified into two sets: service checkpoint image and delta checkpoint image. The identical parts of all VMs that provide the same service are checkpointed once as the service checkpoint image. All VMs that provide the same service can share with each other the service checkpoint image. The remaining checkpoint images, the delta checkpoint images, only save the modified page. Second, the checkpoint image storage issue is modeled as an optimization problem. Finally, the optimization problem is divided into two sub-problems: storage node selection and routing path selection. In addition, we also present an efficient heuristic algorithm to solve these sub-problems. Based on characteristics of the data center network and server failure predictor, the algorithm tried to minimize network resource consumption. To verify the effectiveness of our approach, we extend Cloudsim, a well-known cloud simulator, to a new simulator FTCloudsim by adding fat-tree data center network construction modules, failure and repair event trigger modules, checkpoint-based service recovery modules, and so on. We implement all approaches in FTCloudsim and compare RADS-CKP with the other approaches on total task execution time, average lost time, total network resource consumption, total storage space usage, and other performance metrics. Experimental results show that our proposed approach can reduce network resources and storage resources consumption while still guaranteeing cloud service reliability.

The rest of this paper is organized as follows. In Section 2, we review related work. In Section 3, background is presented. We introduce the motivation and technical details of our proposed approach in Section 4. In Section 5, the experimental results are outlined, and we conclude the paper in Section 6.

2 Related Work and Discussion

Cloud service reliability enhancement is an important research problem in cloud computing. Although reliability enhancement and various fault tolerance techniques have been widely studied in distributed systems and high-performance computing, cloud computing and the special architecture of data center network bring new challenge to the research.

To address these challenges, researchers have outlined service reliability problem details that are particular to the cloud environment. Basic cloud computing design philosophies have been shown by [18], [19]. Special factors that can influence cloud service reliability have been focused on by [20], [21]. Lin et al. [22] have presented the data availability assurance problem in case of the data node failure. Bilal et al. [23] have presented a new procedure to quantify the robustness of data center network to failures. Because of the complexity of the cloud environment, modeling cloud service reliability is a critical but difficult problem. Ghosh et al. [24] analyzed the cloud system and systematically evaluated the availability and reliability of cloud service.

Some cloud service reliability enhancement approaches [6], [8], [9] have also been proposed by researchers.

When a host server crashes, VMs hosted on it will go down. Therefore, the services delivered by the VMs will be interrupted. The greater the number of VMs that provide the same service are hosted on the same server, the more serious the damage a failure causes. To address this issue, the work [8], [9] proposed a redundant VM placement approach for multiple applications. The approach serves to ensure that all cloud services can be maintained while any k host servers fail at the same time. In [6], all VMs and their backups that together provide a workflow service are referred to as a “survivable virtual infrastructure”. A mapping algorithm is proposed to map the group to the physical data center. In addition to determining how to map each node to a VM, the algorithm shows how to reserve bandwidth for traffic between each service providing VM and its backup VMs, and between each backup VM and all service providing VMs.

Moreover, in the cloud computing environment, in addition to ensuring cloud service reliability, the cloud service reliability enhancement approach should reduce resource consumption as much as possible based on data center characteristics. Because of the large costs incurred by the replication mechanism, the approach based on it is only suitable for the critical task. To overcome the problem, notable approaches were proposed in [10], [11], [12] to identify the significant part of the complex task to reduce the implementation cost. These approaches first calculate the significance value of each sub-task according to the invocation structures and frequencies. Then, they rank the sub-tasks according to the significance value and determine the redundancy of each sub-task based on it. Unlike the fixed redundancy level approach, Jung et al. [13] reduced the implementation cost by changing the redundancy of a component when a failure occurs.

Even after the above mentioned improvement [10], [11], [12], [13], the implementation of replication mechanism remains costly. Therefore, such a mechanism is more suitable for a real-time task or critical task. However, for some non real-time large-scale tasks, the checkpoint is a relatively more effective approach. If the checkpoint image is stored in the service providing node and then the service providing node crashes, the checkpoint image will become inaccessible. For this reason, the checkpoint images must be periodically transferred to the persistent storage node. But because the data center network resource is limited, the checkpoint traffic may congest the network and affect other cloud services. To solve this problem, Zhang et al. [15] proposed a theoretical delta-checkpoint approach. Moreover, Goiri et al. [16] implemented the approach in [15] and presented a smart checkpoint infrastructure, which only saves the base system once the first checkpoint is complete. The next checkpoint image then only contains the pages modified since the last checkpoint was created.

Nevertheless, these approaches [15], [16] overlook two important factors. For one, the system checkpoint images of VMs that provide the same service to some extent are similar since the base system, ram disk content, and code page are the same among the VMs. Second, the core switches are the bottleneck of the data center network. When the checkpoint images are stored in central storage servers, the checkpoint traffic may congest the core switches. Limrungsi et al [17] proposed a distributed delta-checkpointing approach. The approach stores the checkpoint images on the neighboring host servers. When the service-providing server and image-storage server sharing the same aggregation switches, the checkpoint images may only require transferring by the aggregation switches and edge switches. However, if the checkpoint image-storage server crashes, the checkpoint image will become inaccessible. Moreover, the approach in [17] cannot take full advantage of the two factors.

Different from all the above work, our proposed approach can optimize the cloud service reliability enhancement approach by using checkpoint image sharing and a failure predictor. Taking advantage of the first factor, we reduce the storage resource usage through checkpoint image sharing. Taking advantage of the second factor and selecting the checkpoint image storage node by using the node failure distribution [25], [26], [27], we reduce the network resource and storage resource usage.

3 Preliminaries

To effectively outline the proposed approach, we first introduce the background. We begin this section by intro ducing the data center network architecture. Some basic knowledge of checkpointing is also provided. Note that the notations in Table 1 will be used throughout the paper.

TABLE 1
NOTATIONS

Symbol	Meaning
PM_i	The *i th* physical machine or host server in the data center, *i* = 1, 2, …
CM	Vector of maximum disk size; *CM[i]* stores the maximum disk size of PM_i
BM	Vector of remaining disk size; *BM[i]* stores the remaining disk size of PM_i
$SP(PM_i)$	Selection preference of PM_i
VM_{ij}	The *j th* virtual machine hosted on PM_i
$Switch_x$	The *x th* switch in the data center, *x* = 1, 2, …
$Node_m$	The *m th* node of the data center network. A node can be a host server or a switch. *m* = 1, 2, …
L_{mn}	The link that connects $Node_m$ and $Node_n$
S_k	The *k th* service in the data center, *k* = 1, 2, …
T_l	The *l th* task submitted by the end user, *l* = 1, 2, …
$Con\ (S_k)$	Concurrency of S_k, which denotes the number of virtual machines that provide S_k
$SerImg_k$	Service checkpoint image of S_k
$SImg_k$	System checkpoint image of S_k
$DImg_{kt}$	Delta checkpoint images that are generated by the *t th* VM providing S_k
Size(image)	Function that returns the size of a checkpoint image
PM(image)	Function that returns all servers storing *image*
$pod(SerImg_k)$	Vector. *pod(SerImgk)*[*i*] equals 1 if a copy of $SerImg_k$ is stored in a host server in the *i*th pod
Store(image)	Vector. *Store(image)*[*i*] equals 1 if a copy of *image* is stored in PM_i; otherwise it is 0
$Flow_{link}(image)$	Vector. $Flow_{link}(image)$[*m*, *n*] equals 1 if *image is* routed through L_{mn}; otherwise it is 0
$Flow_{switch}(image)$	Vector. *Flow switch(image)* [*x*] equals 1 if *image* is routed through $Switch_x$; otherwise it is 0
$E(\bullet)$	Function that returns the mean value

3.1 Data Center Network

As shown in Fig. 1, a current data center network typically consists of three-level trees of switches [30], [31], [32]. The top layer is the core tier; the switches in this layer are core switches. The middle layer is the aggregation layer; the switches in this layer are aggregation switches. The bottom layer is the edge layer; its switches are edge switches. The host servers physically attach to the network by connecting to an edge switch. A host server can simultaneously host one or more VMs. All host servers that connect to the same edge switch are called in the same subnet. All host servers that share the same aggregation switches are called in the same pod. The link that connects a core switch and an aggregation switch is a core link, while the link connecting an aggregation switch and edge switch is an aggregation link. The link that connects an edge switch and a host server is an edge link. All traffic moving outside the cloud data center should be routed through the core switch [33], [34]; consequently, the core link easily becomes congested. Hence, how to reduce the network resource comsumption of core link becomes an important problem.

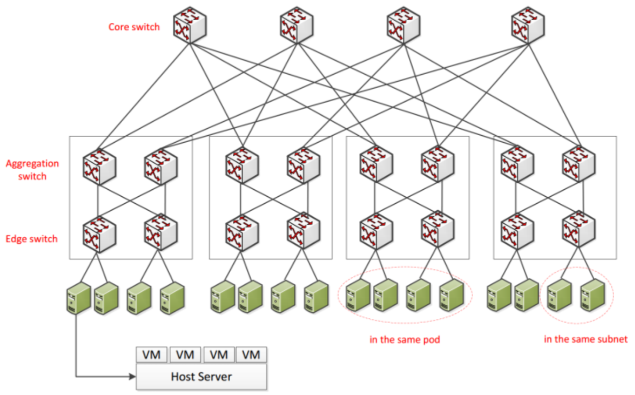

Fig. 1. Fat-tree data center network.

3. 2 Basic Checkpoint Concept

Checkpointing is a conventional technique for enhancing reliability. As shown in Fig. 2, the mechanism periodically saves the current state of a task running on the VM as a checkpoint image. In the event of a host server failure, the task can be resumed from the last saved checkpoint image on another VM.

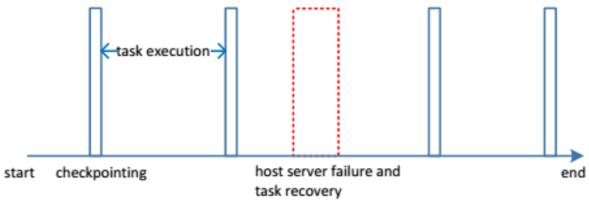

Fig. 2. Task execution process with checkpointing mechanism.

Suppose task T_i is being processed by VM_j, and VM_j is a host on server PM_k. When the checkpointing technique is adopted, the total lost time brought about by a failure of PM_k to T_i can be calculated by the following:

$$t_{lost}(T_i) = (t_{failure} - t_{avail-ckp}) + (t_{recovery} - t_{failure}) \tag{1}$$

where $t_{failure}$ denotes the time when the host server fails, $t_{avail-ckp}$ denotes the time when the current, newest accessible checkpoint image is generated, and $t_{recovery}$ denotes the time when the service resumes. ($t_{failure}$-$t_{avail-ckp}$) is effected by the following three factors: (1) The failure of the image storage server: if the task execution server fails when the image storage server fails, the current interrupted task should restart from the beginning. (2) The time when the failure occurs: if the failure occurs when the current checkpoint image is being created, the current interrupted task must be restarted from the last saved state. (3) The transfer delay from the task execution host server to the image storage server: if the failure occurs while the current checkpoint image is being transferred to the storage node, the task should restart from the last saved state. ($t_{recovery}$- $t_{failure}$) is the transfer delay from the image storage server to the recovery server.

To reduce the impact of the failure, we must minimize $t_{lost}(T_i)$ to maximize cloud service reliability.

4. Motivation and Proposed Approach

In this section, we first introduce the motivation of our approach. The detail of our approach is also provided in this section.

4.1 Motivation

Storage resources must be consumed to persistently store the checkpoint image and network resources for transferring the images to the storage server. The resource usage of different checkpoint approaches is quite different and will be discussed in this section. The overview is provided in two aspects: checkpoint image generation and checkpoint image storage node selection.

4.1.1 Checkpoint Image Generation

A checkpoint image can be a gigabyte size. As mentioned, data center network resources are shared by all host servers or VMs in the cloud data center. The voluminous checkpoint traffic may consume a significant amount of network resources and congest the cloud data center. Regular cloud services may be greatly affected. Work [15], [16] observed that only a small percentage of data has been changed compared with the last checkpoint image. Consequently, they proposed two types of checkpoint images in their approach: system checkpoint image and delta checkpoint image. The checkpoint image containing the base system is called the *system checkpoint image*. The remaining checkpoint images are called *delta checkpoint image*; they contain only the modified page with respect to the last checkpoint image. The delta checkpoint image is periodically generated and transferred. This approach reduces network resources used to transfer the image to the central storage server.

In a typical scenario, the cloud service provider will simultaneously receive a high number of cloud service requests. To complete all service requests in time, the provider will employ several VMs. The requests are scheduled to the VMs according to a certain schedule strategy. However, currently, the checkpoint image is generated and saved in a VM unit [15], [16], [17]. In other words, current approaches generate and save a copy of the system checkpoint image for each VM that provides the same service. Therefore, the total size of checkpoint images saved for a service can be calculated by the following:

$$CkpSize(S_k) = \sum_{t=1}^{Con(S_k)} (Size(SImg_{kt}) + Size(DImg_{kt})) \tag{2}$$

Some similarity exists among the system checkpoint images of VMs that provide the same service. Among those

VMs, the base system, RAM disk content, kernel, configuration file, code page, and static data page are the same. Therefore, saving a copy of the system image for each VM is not required. We refer to the checkpoint image that can be shared by all VMs providing the same service as the *service checkpoint image*, which is a special system checkpoint image. All other images are also *delta checkpoint images*. Therefore, the total size of checkpoint images which is saved for a service can be calculated by the following:

$$CkpSize(S_k) = \sum_{i=1}^{SS_k} Size(SerImg_i) + \sum_{t=1}^{Con(S_k)} Size(DImg_{kt}) \quad (3)$$

where SS_k (called service checkpoint image storage degree) denotes the number of service checkpoint image copies stored for S_k. SS_k is always much smaller than the service concurrency, which we will discuss in Section 5.4. Therefore, we can reduce the storage resource usage. We will discuss where the checkpoint images are stored in the next section.

4.1.2 Storage Node Selection

The checkpoint image cannot be stored in the service-providing host server since the checkpoint image will become inaccessible when the host server crashes. The checkpoint images are therefore typically stored in the central storage servers. When the images are stored in the central storage servers, checkpoint traffic must still be routed by the core switches. These switches play the role of exchanging outside data for host servers in the data center. The core switches become the bottleneck resulting from the vast amount of data flow. To reduce the checkpoint traffic which may congest the core switches, work [17] proposed a distributed checkpointing approach. In this approach, the VM images are saved not on the central storage servers but on neighboring host servers. If the service-providing server and image storage server are in the same pod, the checkpoint traffic must be routed only by the aggregation switches and edge switches. If the service-providing server and image storage server are in the same subnet, the checkpoint traffic must be routed only by the edge switches.

A major flaw of this approach [17] is that host server crashes may make checkpoint image inaccessible. Many researchers have analyzed the logs of large-scale systems. They have shown that the failure event of host servers has a certain relationship with time and space. Therefore, we can employ these results as a failure predictor and select the storage node based on the following rules.

Rule 1 (Space distribution rule) The researchers have found that a machine failure event shows a strong space limitation [25], [26], [27], [28]. The failures are not uniformly distributed among all host servers. Most of the failures occur in a small fraction of servers. The failure events hit a host server successively after the first failure. A host server that has recently failed has a greater chance of failing in the future. The higher the recent number of successive failure times *f* is, the node is more prone to be an easy to failure host server recently. We can therefore model the selection preference of a server based on *f*. We refer to it as node selection preference on space, which is denoted by $SP_s(f)$. $SP_s(f)$ is modeled by using a monotone decreasing function as follows:

$$SP_s(f) = a - b * \exp(f / c) \quad (4)$$

where *f* is the number of recent failure times of the host server, "recent" is calculated by f_{max}*T_f , where f_{max} is the max number of the successive failure among all host servers, T_f is the average inter-arrival time between successive failures; *a* is a number, *b* and *c* must be positive numbers to ensure the function is a monotone decreasing function. The value selection of *a*, *b* and *c* must satisfy some comstraints, which we will discuss later.

Rule 2 (Time distribution rule) In reliability theory [36], the failure inter-arrival time of a machine consistently satisfies a distribution. The studies [27], [28], [29] observed that the distribution of inter-arrival times between failures of a host server is well modeled by a Weibull distribution. The probability density function (pdf) and cumulative distribution function (cdf) of the Weibull distribution are as follows [35]:

$$pdf_{Weibull}(t) = (\frac{shape}{scale}) \times (\frac{t}{scale})^{shape\text{-}1} \times e^{-(\frac{t}{scale})^{shape}} \quad (5)$$

$$cdf_{Weibull}(t) = 1\text{-}(e^{-(\frac{t}{scale})^{shape}}) \quad (6)$$

where *shape* affects the shape of the distribution, and *scale* affects the statistical dispersion of the probability distribution. In these studies [27], [28], [29], *shape* is between 0 and 1, *scale* is positive. Although the value of *scale* and *shape* in these studies are different, the virtual resource provider can calculate the actual value based on its own log.

The hazard function [36] is used in reliability theory to represent the probability that the system will fail in a specified time given no failure before time *t*. The hazard function of Weibull distribution is as follows:

$$h(t) = \frac{pdf_{weibull}(t)}{1 - cdf_{weibull}(t)} = \frac{shape}{scale}(\frac{t}{scale})^{shape-1} \quad (7)$$

With a greater the value, the host server tends to fail in the future. We model the node selection preference on time of a special host server by the following:

$$SP_t(t) = 1 - h(t) = 1 - \frac{shape}{scale}(\frac{t}{scale})^{shape-1} \quad (8)$$

Therefore, the greater the value is, the server tends to work well in the future. 1- *h*(*t*) is an ascending function when parameter shape is smaller than 1. If 1- $h(t_i)$>1- $h(t_j)$ for PM_i and PM_j currently, 1-$h(t_i + \Delta t)$>1-$h(t_j + \Delta t)$. Therefore, we can choose the best host server based on 1-*h*(*t*). Because it takes a period of time to start the host server, *t* is at least greater than 1 s.

When sorting the host servers based on $SP_s(f)$ and $SP_t(t)$, we need to compare twice between any two host servers. To obtain the selection preference by a single value and reduce the number of comparisons, we model the selection preference of a host server by the following:

$$SP(f,t) = [1 + \mathrm{sgn}(-f) + (1 - \frac{shape}{scale}(\frac{t}{scale})^{shape-1})] + [\mathrm{sgn}(f\text{-E}(f)) + 1] * [a - b * \exp(f / c)] \quad (9)$$

The value selection of a, b and c can affect the combination, which we will discuss later in proof 1.

Proposition 1. *If we properly select the value of a, b, c, and the storage node selection of the checkpointing mechanism selects the storage node by sorting the host servers based on $SP(f,t)$, the solution will be optimal based on both the space distribution rule and the time distribution rule. In other words, based on $SP(f,t)$, the storage node selection can select the node that never fails recently, and avoid selecting the node with a very high failure frequency. If several host servers all have low failure frequency, the selection will select the one with the largest node selection preference on time.*

Proof 1. Firstly, we need show that if several host servers all have low number of failure times, the selection will select the one with the largest $SP_t(t)$. We can obtain that $SP(f_i,t_i)$ equals $SP_t(t_i)$ when $0 < f_i < E(f)$, and therefore, $SP(f_i,t_i)$ is determined by $SP_t(t)$. The larger $SP_t(t_i)$ is, the higher $SP(f_i,t_i)$ is.

Then, we need make sure that the selection can select the node that never fails recently, and avoid selecting the node with a very high number of failure times. Clearly, R is a segmented function. Because *scale* is larger than 1 s in all studies, the range of $SP_t(t)$ is (0, 1]. Therefore, the range of $SP(f,t)$ are (1,2], (0,1], $[0+(a-b*\exp(E(f)/c))$, $1+(a-b*\exp(E(f)/c))]$, and $[-\infty, 1+2(a-b*\exp(E(f)/c))]$ when f belongs to [0, 0], $(0, E(f))$, $[E(f), E(f)]$, and $(E(f), \infty)$ respectly. We now need to make sure that if $f_i < f_j$, and f_i, f_j belong to different "segment", then $\min(SP(f_i,t_i)) > \max(SP(f_j,t_j))$. Therefore, a and b must satisify:

$$(1+a) < b*\ \exp(E(f)/c) \tag{10}$$

Now, we must show that the $SP(f_i,t_i)$ fall quickly when the frequency is high. In other respect, if $f_i > f_j > E(f), \forall t_i, t_j$, then $SF(f_i,t_i) \le SF(f_j,t_j)$; therefore, $SP_t(t_i)+2*[a-b*\exp(f_i/c)] \le SP_t(t_j)+2*[a-b*\exp(f_j/c)]$. Consequently, we must show that $\max(SP_t(t_i))+2[a-b*\exp(f_i/c)] \le \min(SP_t(t_j))+2[a-b*\exp(f_j/c)]$. For $\max(SP_t(t))=1$, and $\min(SP_t(t))=0$, we can obtain the format that $1+2[a-b*\exp(f_i/c)] \le 0+2[a-b*\exp(f_j/c)]$, and then $1-2b*\exp(f_i/c) \le -2b*\exp(f_j/c)$.

Suppose that $f_1 < f_2 ... < f_n$, we let $df_x = f_x - f_{x-1}$, $dSP_s(f_x) = SP_s(f_x) - SP_s(f_{x-1})$. Firstly, note that:

$$\frac{\partial^2 SF_s(f)}{\partial f^2} = -(b/c^2)\exp(f/c) < 0 \tag{11}$$

Then, $SP_s(f)$ is a convex function. Suppose that $\forall x \forall y$, $f_x - f_{x-1} = f_y - f_{y-1}$. For $SP_s(f)$ is a convex function, we can obtain the following:

$$dSP_s(f_1) < dSP_s(f_2) .... dSP_s(f_n) \tag{12}$$

Moreover, we note that:

$$\frac{dSP_s(f)}{df} = -(b/c)*\exp(f/c) < 0,$$

Then, $SP_s(f)$ is a decreasing function. We can obtain the following:

$$dSP_s(f_x) < dSP_s(f_y) \ \ when\ df_x < df_y \tag{13}$$

Based on (10), (11), and (12), we must show that $1-2b*\exp(f_i/c) \le -2b*\exp(f_j/c)$ when $f_i = 1, f_j = 0$. In other words, we must show the following:

$$1+2b < 2b*\exp(1/c) \tag{14}$$

Therefore, if a, b, and c satisfy (10) and (14), the storage node selection can obtain the solution that is optimal based on Rule 1 and Rule 2. We choose a = -5, b = 10, c = 10. As we can see, $1-5<0$, $10*\exp(E(f)/10)>0$, and $1+20<20*\exp(1/10)$. **(End Proof)**.

If a checkpoint image is stored in a host server PM_i, the storage reliability of the checkpoint image is proportional to the selection preference of the host server. Then, the storage reliability of the checkpoint image is modeled as follows:

$$SR(img) = SP(f,t) \tag{15}$$

We will select the storage node based on (15) in our approach.

4.2 Proposed Approach

The objective of our approach is to select the storage node and routing path for the checkpoint image. We intend to minimize network and storage resource consumption based on the available information. In this paper, unless otherwise specified, the checkpoint image generation host server is called the source node of the checkpoint image.

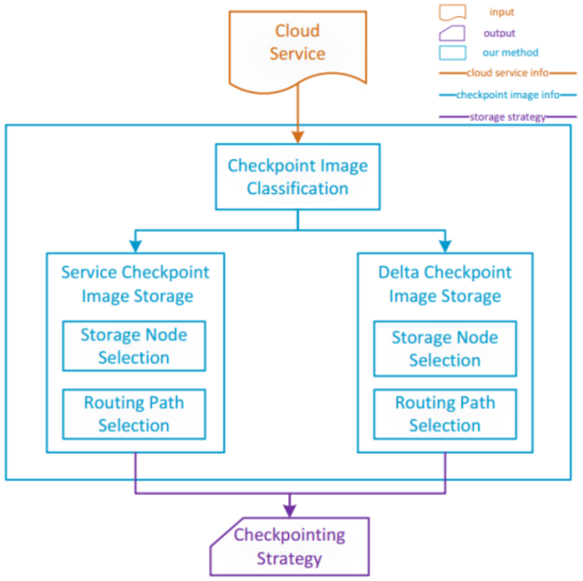

Fig. 3. RADS-CKP architecture.

Fig. 3 depicts the system architecture of our approach, which includes the following three modules.

1. Checkpoint image classification. The module divides the checkpoint image into two sets: service checkpoint images and delta images. Note that a service image can be shared by all VMs that pro-

vide the same service.

2. Service checkpoint image storage. The module selects the optimal storage node and routing path for the service image.
3. Delta checkpoint image storage. The module selects the optimal storage node and routing path for the delta checkpoint image.

We will consider the image storage reliability and the network congestion when selecting the storage node and transfer path. The details of our approach will be introduced in Sections 4.2.1 to 4.2.3.

4.2.1 Checkpoint image classification

We first discuss creation of the service and delta images. In traditional approaches, the system checkpoint image is generated after an initial time offset. Therefore, the task context, memory content, and disk content are not the same among the VMs when checkpointing, and the system checkpoint image cannot be shared among the VMs. In a cloud data center, the VMs are created from VM images [38], [39]. Therefore, the virtual resource provider can create a checkpoint image containing all application programs required by a cloud service provider and maintain the image in the database. It services as a service checkpoint image. All VMs providing the service will be started from the service checkpoint image. The virtual resource provider can readily perform this task.

The existing delta checkpointing approaches [15], [16] add a bit table for each VM to indicate which memory page or disk page has been modified since the last checkpointing. The bit table is cleared after each checkpointing. To support the service checkpoint image in our approach, another new bit must be added to the table to indicate which page has been modified with respect to the service image. The bit table is only used when a delta image storage node crashes. We must re-generate the delta image, which contains the modified page compared with the service checkpoint image. Because it only requires one bit to indicate one page, the bit table uses minimal space.

4.2.2 Service checkpoint image storage

This section describes how to select the optimal storage node and routing path for the service checkpoint image. Given the importance of the service checkpoint image, our approach stores a copy of the service image in the central storage server. The size of the service checkpoint image is relatively large; it may therefore consume too much time to transfer a service image from the storage node to the recovery node in the recovery stage. Considering the tradeoff between the recovery time and disk usage, we therefore store a copy of the service checkpoint image in each pod in an adjoining style. To start or recover the service, our approach must transfer a copy of the service image to the target service-providing server. We copy one image from the service-providing server to a host server in the same pod if there is currently no service checkpoint image in the pod. Consequently, the transfer will not use the core link, and our approach can thereby reduce the recovery time if the recovery node is in this pod the next time.

The storage node and routing path selection problem can be formulated as the following optimization problem:

$$Max\,USP(SerImg_k)\,and\,Min\,UL(SerImg_k) \tag{16}$$

Subject to:

$$\begin{aligned} UL(SerImg_k) = \sum_i \sum_j Flow_{link}(SerImg_k)[i,j]*delay(L_{ij}) \\ +\sum_i Flow_{switch}(SerImg_i)[i]*delay(switch_i) \end{aligned} \tag{17}$$

$$USP(SerImg_k) = SP(SerImg_k) \tag{18}$$

$$pod(SerImg_k)[x] = 1,\ 1 \le x \le max\,pod\,num \tag{19}$$

$$PM(S_k) \cap PM(SerImg_k) = null \tag{20}$$

$$PM(SerImg_k) \cap PM(DImg_{kt}) = null \tag{21}$$

$$BM + size(SerImg_k) \times Store(SerImg_k) \leqslant CM \tag{22}$$

$$\sum_j Flow_{link}(SerImg_k)[i,j] - \sum_n Flow_{link}(SerImg_k)[n,i] = \begin{cases} 1 & if\ node_i\ is\ the\ generation\ node\ of\ SerImg_k \\ 0 & otherwise \\ -1 & if\ node_i\ is\ the\ storage\ node\ of\ SerImg_k \end{cases} \tag{23}$$

where the objective function is to maximize *SP*, while the minor objective function is to minimize the storage and network resources that the approach consumes. The constraint in (19) ensures that each pod stores a copy of the service image. The constraint in (20) indicates that a service image cannot be stored in a node that provides the service. The constraint in (21) specifies that the service image and delta image cannot be stored in the same node. The constraint in (22) indicates that the size of the service image should not exceed the spare disk space. The constraint in (23) specifies that we need to find a transfer path for the service checkpoint image.

The above optimization problem can be divided into two sub-problems: storage node selection and routing path selection. However, there are a huge number of host servers, switches and links in the data center; therefore, the possible solutions are exponentially large. We consequently must find a subset of good host servers and search for the best solutions from them. As we know, information exchanged between hosts in the same subnet must only be transferred by an edge switch. When two hosts are in the same pod, all communicated traffic must be routed through the edge switches and aggregation switches. Therefore, the delay becomes greater. If the two host servers are in different pods, the delay becomes even larger. Therefore, we first verify the nodes in the subnet against the checkpoint image generating node. If no node in the same subnet satisfies all constraints, we then search the storage node in the same pod.

Algorithm 1 describes the details of our storage node and routing path selection algorithm. First, we sort all hosts in the same subnet with the source node by the selection preference. We then traverse the node list. If free space of the current node is larger than the service image size, and if it also satisfies all other constraints, we select it as the storage node. Otherwise, we sort all host

Algorithm 1: Service checkpoint image storage node and routing path selection

```
Input: host server $PM_s$ that $SerImg_k$ is fetched from,
       $subnet_s$ that $PM_s$ belongs to, $pod_s$ that $PM_s$ belongs to
Output: Service image storage server $storageserver$, and the
        image transfer path $path$
$storageserver$ = Storage node selection($PM_s$, $SerImg_k$, $subnet_s$, $pod_s$);
if $storageserver \neq null$ then
    select a path from $storageserver$ to $PM_s$ and assign the path to $path$;
end
final ;
return $storageserver$ and $path$;
```

Algorithm 2: Storage node selection

```
Input: host server $PM_s$ that the checkpoint image $Img$ is
       fetched from, $subnet_s$ that $PM_s$ belongs to, $pod_s$ that
       $PM_s$ belongs to
Output: Image storage server $storageserver$
for each host server $PM_i$ in the same subnet with $PM_s$ do
    if $PM_i$ is not a service providing node or checkpoint image storage node of $S_k$ then
        add $PM_i$ to $candidateList$ ;
    end
end
sort $candidateList$ by reliability desc;
init $storageserver$ ; for each $PM_k$ in $candidateList$ do
    if $SP(PM_k) \geq E(SP)$ of $pod_i$ and $BM_k \leq$ size of $Img$ then
        assign $PM_k$ to $storageserver$;
        goto final;
    end
end
clear $candidateList$;
add all other subnets in $pod_s$ to $netList$;
for each subnet $subnet_j$ in $netList$ do
    clear $candidateList$;
    for each $PM_i$ in $subnet_j$ do
        if $PM_i$ is not a service providing node or checkpoint image storage node of $S_k$ then
            add $PM_i$ to $candidateList$;
        end
    end
    sort all host in $candidateList$ by reliability desc;
    for each $PM_k$ in $candidateList$ do
        if $SP(PM_k) \geq E(SP)$ of $pod_i$ and $BM_k \leq$ size of $Img$ then
            assign $PM_k$ to $storageserver$ ;
            goto final;
        end
    end
end
final ;
return $storageserver$;
```

servers in the same pod with the source node by the selection preference. We then traverse the list and ensure the current node satisfies all constraints. The first node that satisfies all constraints will be selected as the storage node. Each time the service checkpoint image only need to be saved once. Therefore, the transfer will have little effect on the network. We randomly select a path for the transfer to reduce the time cost.

4.2.3 *Delta checkpoint image storage*

Storage node and routing path selection problem for the delta image can be formulated as the following optimization problem:

$$Max\, USP(DImg_{kt})\, and\, Min\, UL(DImg_{kt}) \tag{24}$$

Subject to:

$$UL(DImg_{kt}) = \sum_i \sum_j Flow_{link}(DImg_{kt})[i,j] * delay(L_{ij}) + \sum_i Flow_{switch}(DImg_{kt})[i] * delay(switch_i) \tag{25}$$

$$USP(DImg_{kt}) = SP(DImg_{kt}) \tag{26}$$

$$PM(S_k) \cap PM(DImg_{kt}) = null \tag{27}$$

$$PM(SerImg_k) \cap PM(DImg_{kt}) = null \tag{28}$$

$$\bigcap_{i=1}^{con(s_k)} PM(DImg_{ki}) = null \tag{29}$$

$$BM + size(DImg_{kt}) \times Store(DImg_{kt}) \leqslant CM \tag{30}$$

$$\sum_j Flow_{link}(DImg_{kt})[i,j] - \sum_n Flow_{link}(DImg_{kt})[n,i] = \begin{cases} 1 & if\ node_i\ is\ the\ generation\ node\ of\ DImg_{kt} \\ 0 & otherwise \\ -1 & if\ node_i\ is\ the\ storage\ node\ of\ DImg_{kt} \end{cases} \tag{31}$$

where the objective function of the optimization problem is to maximize the selection preference, while the minor objective function is to minimize the storage and network resources that the approach consumes. The constraint in (27) indicates that a service image cannot be stored in a node that provides the service. The constraint in (28) specifies that the service image and delta image cannot be stored in the same node. The constraint in (29) indicates that delta checkpoint images of VMs providing the same service cannot be stored in the same node. The constraint in (30) indicates that the size of the service image should not exceed the spare disk space. *size(DImg$_{kt}$)* grows with the increase of the checkpoint interval. We model the size of *DImg$_{kt}$* according to [16], [17]. The constraint in (31) specifies that we need find a transfer path for the delta chekpoint image.

Algorithm 3 describes the details of our storage node and routing path selection algorithm. The core of the algorithm is the same as the one outlined in Section 4.2. However, the delta images must be periodically generated and transferred; an optimal routing path must be identified for the checkpoint traffic. First, the storage node in the same subnet as the source node is searched. If no node in the same subnet satisfies all constraints, all host servers in the same pod are added to the candidate list.

The candidate list is then sorted by the *SP*. At that point, the list is traversed and the current node is verified in terms of satisfying all constraints. If a node satisfies all constraints, the shortest path is then sought using the Dijkstra algorithm [37] to balance the load. If there is no node and path that satisfies all constraints, the delta check point image is stored in the central storage server.

If a delta image storage node crashes, the new node must be searched and the delta image regenerated. The

Algorithm 3: Delta checkpoint image storage node and routing path selection

```
Input: host server $PM_s$ that generates the delta checkpoint
       image $DImg_{kt}$, $subnet_s$ that $PM_s$ belongs to, $pod_s$
       that $PM_s$ belongs to
Output: Delta image storage server $storageserver$,and the
        image transfer path $Path$
1 $storageserver$ = Storage node selection($PM_s$,
  $DImg_{kt}$,$subnet_s$,$pod_s$);
2 if $storageserver \equiv null$ then
3 |   the delta checkpoint image is stored in the central
  |   storage server;
4 |   goto final;
5 end
6 construct weighted topological graph $graph_s$ of $pod_s$;
7 calculate the shortest path from $storageserver$ to $PM_s$ in
  $graph_s$ by using the Dijkstra algorithm;
8 final ;
9 return $storageserver$ and $path$;
```

newly generated delta image contains the modified page comparison of the service checkpoint image.The reselection algorithm is the same as Algorithm 2.

5. EXPERIMENTS

To verify the effectiveness of our approach RADS-CKP, we extend Cloudsim for our experiments. The following sections outline the experimental setting. We then compare the proposed approach with other approaches in terms of total execution time, average lost time, disk usage, and other performance metrics. Finally, we study the parameters of our approach.

5.1 Cloudsim Extension

Cloudsim [40], [41] is a renowned extensible simulation framework that supports modeling of virtual resource allocation, service scheduling, and other functions. We extend Cloudsim as FTCloudsim and add the following modules to support our experiments. More details are in our Demo[1] [42] about FTCloudsim.

- *Fat-tree data center network construction.* A fat-tree data center network is constructed to connect the host servers.
- *Failure and repair event triggering.* Host failure and repair events are triggered. An event can be generated according to a specified distribution. The failure event data and the repair event data can be saved to a file so the experiment can be repeated.
- *Checkpoint image generation and storage.* A checkpoint image is periodically generated, transferred, and stored based on a checkpoint mechanism. This module is extensible, and we implement our approach and other existing approaches by an extension.
- *Checkpoint-based service recovery.* A service is recovered from a host failure based on the latest available checkpoint image. If there is no accessible checkpoint image, it restarts the service and reprocesses the interrupted task from the beginning.

5.2 Experimental Setup

We construct a 16-port fat-tree data center network in FTCloudsim. Therefore, there are 64 core switches and 16 pods in the data center. Each pod is comprised of 8 aggregation switches and 8 edge switches. That is, there are 128 aggregation switches and 128 edge switches in the cloud data center. According to [6], the capacity of the core and aggregation links is set as 10 Gps, and the capacity of the edge link is set as 1 Gps. The transfer delays of the core, aggregation, and edge switches are 1 s, 1 s, and 2 s, respectively. Each edge switch can connect to 8 host servers, and each host server can host 4 VMs. Therefore, the data center contains 1024 host servers and 4096 VMs. The VM configuration is based on [16]. The base system is 769 M, the RAM disk is 5.3 M, the kernel is 1.6 M, the memory size is 512 M, and the disk size is 1 G. The memory and disk sizes of each host are 4 G and 100 G, respectively. There are 120 services delivered from the datacenter, and service concurrency is uniformly distributed between 20 and 30. We design 3000 total tasks for all services. The task size is uniformly distributed between 10 and 20 h. All service requests are randomly allocated to a VM that provides the given service. The distribution of failure events is generated according to [25], [26], [27], [28]. In space distribution, 8% of all host servers experience almost all the failure events, the number of successive failure times *f* is uniformly distributed between 1 and 3, the average number of successive failure times is 2. In time distribution, the parameter shape is 0.75, and the parameter scale is 60 h. The checkpoint image information is modeled on [16], [17]. The checkpoint interval is 600s. The checkpoint image size grows with the increase of the checkpoint interval, and the convex function is set as ($143*\log_{10}T- 254$) M, where T is the checkpoint interval. The checkpoint image merge time is 0.9 s. The recovery host server is searched in the center of the image storage node. To study the performance of our approach (RADS-CKP), we compare RADS-CKP with other six competing approaches, which are as follows:

- *Non-CKP.* No reliability enhancement approach is employed.
- *CB-CKP.* The checkpoint image contains all information required to resume the service. All checkpoint images are stored on central storage servers.
- *CD-CKP.* This approach is proposed by [16]. The checkpoint image is classified into system checkpoint image and delta checkpoint image. All checkpoint images are stored on central storage servers.
- *RDD-CKP.* This is a distributed checkpoint image storage approach. The checkpoint images are classified into system checkpoint and delta checkpoint images. It randomly selects the storage node.
- *ODD-CKP.* This approach is proposed by [17]. This is a distributed checkpoint image storage approach. The checkpoint images are classified into system checkpoint image and delta checkpoint image. Additionally, the storage node is selected based on the network characteristic.
- *RADD-CKP.* This is a distributed checkpoint image

[1] http://youtu.be/yMyz2gesywA

storage approach. The checkpoint image is classified into the system checkpoint image and delta images. The storage node is selected based on the network characteristic and the failure predictor.

All approaches are evaluated using the following performance metrics:

- *Total execution time*: the total time the approach takes to complete all tasks, which can be calculated by the following:

$$t_{total} = \sum_i (t_{end}(T_i)\text{-}t_{start}(T_i)) \quad (32)$$

where $t_{start}(T_i)$ is the time T_i is submitted, and $t_{end}(T_i)$ is the time T_i is completed.

- *Average lost time*: average lost time because of a host failure, which can be calculated by the following:

$$t_{E(lost)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{down(i)}\sum_{j=1}^{down(i)} t_{lost}(T_j)\right) \quad (33)$$

where n denotes the number of failure events, and $down(i)$ denotes the interrupted task caused by a failure event.

- *Packets processed*: this performance metric consists of four sub-metrics.

  The total size of packets transferred by the root switches, which can be calculated by the following:

$$Packet_{root} = \sum_i X_i \times size(packet_i) \quad (34)$$

where X_i equals the frequency with which $packet_i$ has been transferred by the root switches.

The total size of packets transferred by the aggregation switches, which can be calculated by the following:

$$Packet_{agg} = \sum_i Y_i \times size(packet_i) \quad (35)$$

where Y_i equals the frequency with which $packet_i$ has been transferred by the aggregation switches.

The total size of packets transferred by the edge switches, which can be calculated by the following:

$$Packet_{edge} = \sum_i Z_i \times size(packet_i) \quad (36)$$

where Z_i equals the frequency with which $packet_i$ has been transferred by the edge switches.

The total size of packets transferred by all switches, which can be calculated by the following:

$$Packet_{all} = Packet_{root} + Packet_{agg} + Packet_{edge} \quad (37)$$

We simply count the packets related to the sevice start, restart, and recovery.

- *Total disk usage*: disk usage for the storage of the checkpoint image, which can be calculated by the following:

$$S_{total} = \sum_i imagesize(host_i) \quad (38)$$

where *imagesize(host$_i$)* returns the size of all checkpoint images that are stored on *host$_i$*.

5.3 Performance Comparison

5.3.1 Reliability Enhancement

We first study the performance of reliability enhancement, which is evaluated by the total execution time and average lost time. The results are presented in Figs. 4 and 5. The two figures show that:

- The total execution time and average lost time of Non-CKP are longer than other approaches. This is because all other approaches employ some fault-tolerance mechanism. Therefore, they can reduce the time loss caused by a host server failure.
- Of all approaches, the total execution time and average lost time of distributed approaches are shorter than those of central storage server approaches. This is because when the checkpoint images are stored on the central storage server, the checkpoint traffic should be transferred by all three layer switches. However, when the images are stored in neighboring host servers, the checkpoint traffic only requires transfered by the aggregation and the edge switches. If the storage node and the recovery node are in the same subnet, the checkpoint traffic only need to be transferred through the edge switches. Therefore, it takes less time to transfer the checkpoint image from the storage node to the recovery node, which reduces the total execution time and average lost time.
- The total execution time of all distributed approaches is approximately the same; however, RADS-CKP (our approach) consumes slightly more time than ODD-CKP and RADD-CKP. This is because there is at most one copy of a service checkpoint image in each pod. Therefore, the service checkpoint image must be routed by the aggregation switches and the edge switches in the recovery stage. In ODD-CKP and RADD-CKP, however, the checkpoint image may only require transference by the edge switch if the recovery node and the storage node are in the same subnet. Therefore, its recovery time (7.21 min) is slightly longer than it is for ODD-CKP (7.12 min) and RADD-CKP (7.07 min); nevertheless, the difference is not obvious.

We can learn from the above overview that our approach outperforms all centralized approaches. Moreover, it demonstrates the same effect on service reliability enhancement as other distributed approaches. However, as will be shown, our proposed approach consumes fewer resources than the other approaches.

5.3.2 Network Resource Consumption

The network resource consumption of all approaches is studied. Figs. 6 to 9 present the experimental results. Figs. 3 to 5 provide the results of $Packet_{root}$, $Packet_{agg}$, and $Packet_{edge}$, respectively. Fig. 6 displays the results of $Packet_{all}$, which is the total size of the check point image data processed by all switches in the data center. The results of Non-CKP in Figs. 8 and 9 are not zero but are below the starting value of the y-axis. The figures show that:

- Compared to centralized approaches, the total data processed by the core switches in distributed approaches is much smaller. This is because the distributed approaches store the checkpoint image in the

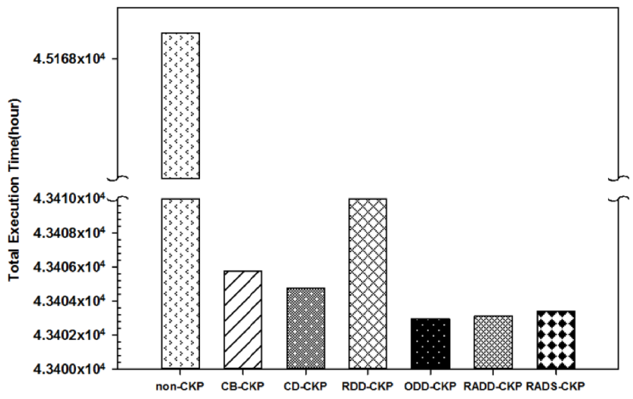

Fig. 4. Total execution time

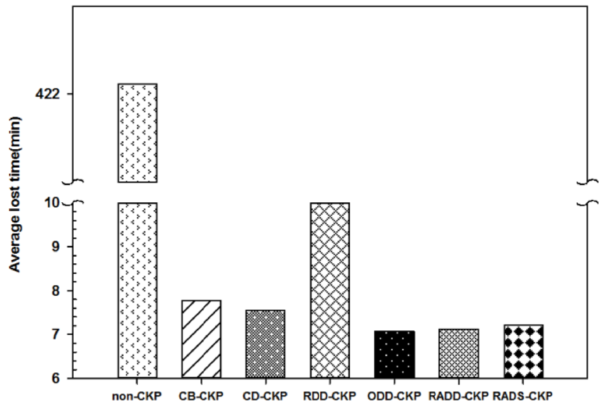

Fig. 5. Average lost time

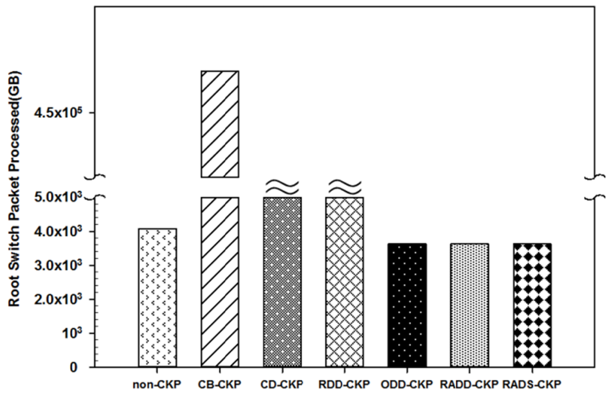

Fig. 6. $Packet_{root}$

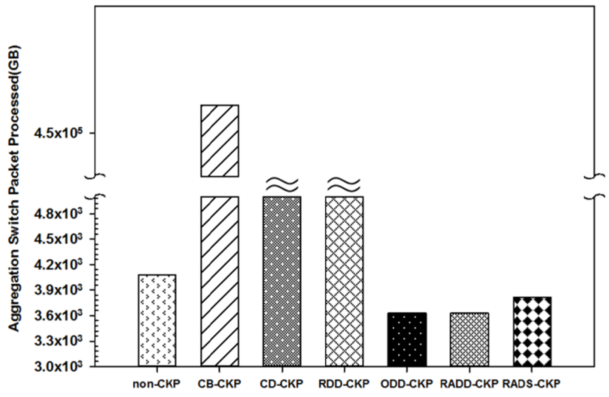

Fig. 7. $Packet_{agg}$

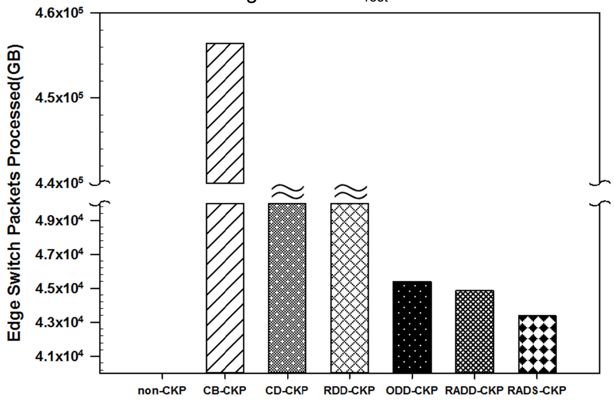

Fig. 8. $Packet_{edge}$

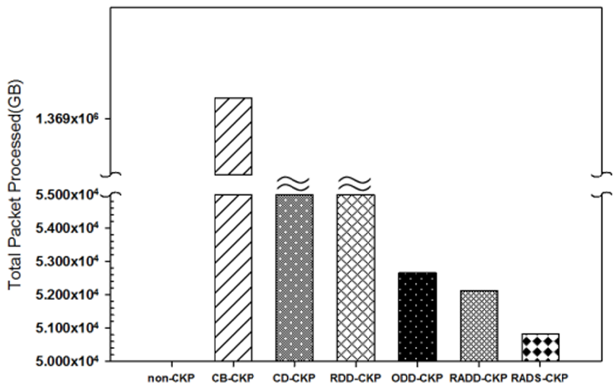

Fig. 9. $Packet_{all}$

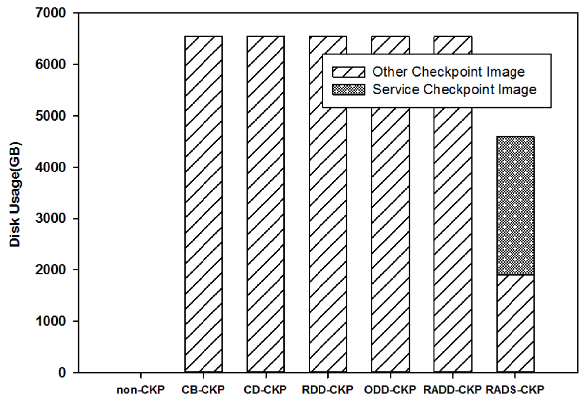

Fig. 10. Total disk usage

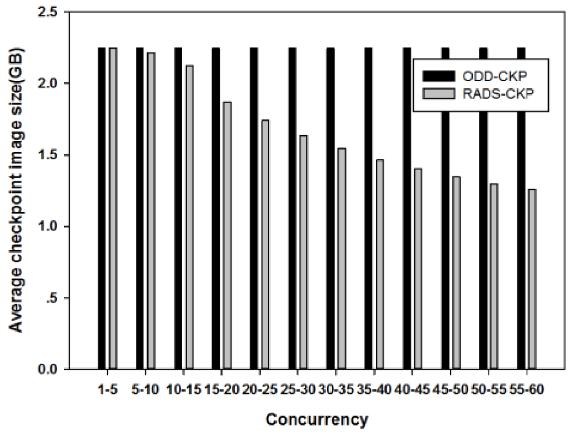

Fig. 11. Average checkpoint image size

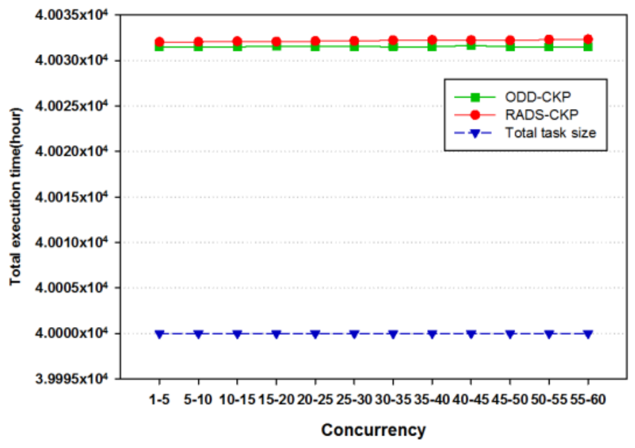

Fig. 12. Total execution time

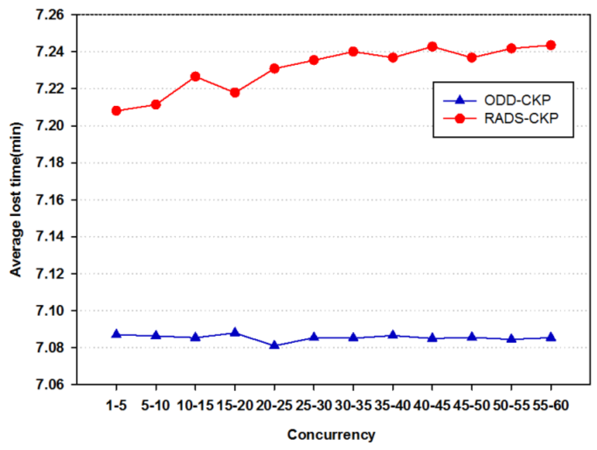

Fig. 13. Average lost time

neighboring host servers. Therefore, the distributed approaches consume less core layer network resources.

- Among the seven approaches, the total data processed by aggregation switches in distributed approaches is much smaller than those in centralized approaches. However, our approach consumes slightly more aggregation layer network resources. This is because the service-providing host and image storage node may be in the same subnet or pod; the checkpoint traffic does not need to traverse the core layer. Therefore, the distributed approaches consume less aggregation layer network resources. We must store a copy of the service checkpoint image in each pod, which would take up some aggregation layer network resources.
- Of all seven approaches, our approach consumes the least edge layer network resources and total network resources. When the checkpoint image is lost because of a host server failure, the system checkpoint image must be retransferred. The data retransmission will require more network resources. We consider the failure chance of the host servers; therefore, there is a smaller chance that a check-point image must be transferred again.

5.3.3 Disk Usage

Fig.10 presents the disk usage of all approaches. Although the approaches consume varying amounts of bandwidth resources, their disk usage is approximately the same. Compared with other approaches, the disk usage of RADS-CKP is much less since all other approaches store one copy of the system checkpoint image for each VM. In the proposed approach, however, the similar part of the system check point image is shared by all VMs that provide the same service. Therefore, our approach can save considerably more storage resources. In the next section, we show how service concurrency impacts performance.

5.4 Impact of Service Concurrency

To study the impact of service concurrency on reliability enhancement and storage resource consumption, we compare ODD-CKP and RADS-CKP with different service concurrency value settings. For comparison, the task size is set as 48000s for all the tasks. Other configurations are the same as in Section 5.3.

5.4.1 Impact on Storage Resource Consumption

The performance metrics is average checkpoint image size, which is calculated by dividing the total image size by the number of service-providing VMs. As shown in Fig. 11, the average checkpoint image size in ODD-CKP is not influenced by service concurrency. However, the average image size decreases with the increase of service concurrency. When the average concurrency is below 5, the average image size is even larger than that of ODD-CKP. This is because a copy of the service checkpoint image is stored in each pod in an adjoining style in addition to a copy being stored in the central storage server. When the service concurrency is smaller than the number of pods, the number of RADS-CKP system checkpoint image copies is larger than that of ODD-CKP. However, with the increase of service concurrency, the service image can be shared by all VMs that provide the same service. The number of RADS-CKP system checkpoint image copies is smaller than for ODD-CKP. Therefore, the average checkpoint image size decreases. If the service concurrency is larger than the number of pods, our approach can save ample disk resources. When the port number increases to 64, there are 65,536 hosts in the data center. When the pod number increases to 128, there are 524,288 servers in the data center. Therefore, it is a common condition that the service concurrency is larger than the pod number.

5.4.2 Impact on Reliability Enhancement

The performance metrics consist of total execution time, average lost time. As shown in Fig. 12, the total execution time of our approach is slightly longer than for ODD-CKP. This is because the average lost time of our approach is longer than for ODD-CKP. In our approach, there is only one copy of a service image in each pod; therefore, the images must be transferred through aggregation switches and edge switches in almost all conditions. Consequently, the average lost time is slightly longer; as shown in Fig. 13, the difference is within 0.2 min. This difference is negligible and can be disregarded. However, as shown in Fig. 13, the average lost time of our approach increases when the service concurrency increases from 1 to 30. The average lost time remains approximately the same when the service concurrency is larger than 30. This is because when the concurrency is increased to the pod number, there is a greater chance that the recovery node and the image storage node are not in the same subnet but are in the same pod. Therefore, the checkpoint images must be transferred by the edge switches and the aggregation switches in the recovery stage. Hence, the delay increases.

6 Conclusion

In this paper, we examine the problem of cloud service reliability enhancement. In our proposed approach, checkpoint images are classified into service checkpoint images and delta checkpoint images. Storage resource usage is reduced through service checkpoint image sharing. The checkpoint image storage node and routing path selection problem are then formulated as an optimization problem. By exploiting the characteristics of the data center network and the host server failure information, we present a heuristic algorithm to efficiently select the optimal storage node and routing path. To verify the effectiveness of our approach, we extended Cloudsim and conduct extensive experiments. The experimental results showed that the proposed approach can guarantee reliability while consuming fewer network and storage resources than other approaches.

Our future work will involve enhancing the reliability of special cloud services, such as mapreduce service and workflow service. We will also consider how provide reliability-differentiated virtual resources for cloud service.

References

[1] M. Armbrust et al., “Above the Clouds: A Berkeley View of Cloud Computing,” Technical Report EECS-2009-28, Electrical Eng. and Computer Science Dept., Univ. of California, Berkeley, 2009.

[2] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Future Generation computer systems*, vol.25, no.6, pp.599-616, 2009.

[3] “An introduction to designing reliable cloud services“, white paper of Microsoft, http://www.microsoft.com/en-us/download/details.aspx?id=34683, 2013.

[4] E. Bauer, and R. Adams, *Reliability and availability of cloud computing*, John Wiley & Sons, 2012.

[5] M. Schwarzkopf, D. G. Murray, and S. Hand, “The seven deadly sins of cloud computing research,” *Proc. the 4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '12)*, Jun. 2012.

[6] J. Xu, J. Tang, K. Kwiat, W. Zhang, and G. Xue, “Survivable virtual infrastructure mapping in virtualized data centers,” *Proc. IEEE 5th Int'l Conf. Cloud Computing (Cloud '12)*, pp. 196-203, Jun. 2012.

[7] Y. Zhang, Z. Zheng, and M.R. Lyu, “BFTCloud: A byzantine fault tolerance framework for voluntary-resource cloud computing,” *Proc. IEEE Int'l Conf. Cloud Computing (Cloud '11)*, pp. 444-451, Jul. 2011.

[8] F. Machida, M. Kawato, and Y. Maeno, “Redundant virtual machine placement for fault-tolerant consolidated server clusters,” *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS'10)*, pp. 32-39, Apr. 2010.

[9] E. Bin, O. Biran, O. Boni, E. Hadad, E.K. Kolodner, Y. Moatti, and D.H. Lorenz, “Guaranteeing high availability goals for virtual machine placement.,” *Proc. 31st Int'l Conf. Distributed Computing Systems (ICDCS' 11)*, pp. 700-709, May. 2011.

[10] Z. Zheng, T. Zhou, M. Lyu, and I. King, “Component ranking for fault-tolerant cloud applications,” *IEEE Transactions on Services Computing*, vol.5, no.4, pp.540-550, 2012.

[11] Z. Zheng, Y. Zhang, and M.R. Lyu, “CloudRank: A QoS-Driven Component Ranking Framework for Cloud Computing,” *Proc. Int'l Symp. Reliable Distributed Systems (SRDS '10)*, pp. 184-193, 2010.

[12] Z. Zheng, T.C. Zhou, M.R. Lyu, and I. King, “FTCloud: A Ranking-Based Framework for Fault Tolerant Cloud Applications,” *Proc. Int'l Symp. Software Reliability Eng. (ISSRE '10)*, pp. 398-407, 2010.

[13] G. Jung, K.R. Joshi, M.A. Hiltunen, R.D. Schlichting, and C. Pu, “Performance and availability aware regeneration for cloud based multitier applications,” *Proc. IEEE/IFIP Int'l Conf. Dependable Systems and Networks (DSN '10)*, pp. 497-506, 2010.

[14] S. Yi, D. Kondo, and A. Andrzejak. “Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud,” *Proc. IEEE 3rd Int'l Conf. Cloud Computing (CLOUD '10)*, pp. 236-243, Jun. 2010.

[15] M. Zhang, H. Jin, X. Shi, and S. Wu. 2010, “Virtcft: A transparent vm-level fault-tolerant system for virtual clusters,” *Proc. IEEE 16th Int'l Conf. Parallel and Distributed Systems (ICPADS '10)*, pp. 147-154, 2010.

[16] Í. Goiri, F. Julia, J. Guitart, and J. Torres, “Checkpoint-based fault-tolerant infrastructure for virtualized service providers,” *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS '10)*, pp. 455-462, 2010.

[17] N. Limrungsi, J. Zhao, Y. Xiang, T. Lan, H.H. Huang, and S. Subramaniam, “Providing reliability as an elastic service in cloud computing,” *Proc. IEEE Int'l Conf. Communications (ICC '12)*, pp. 2912-2917, 2012.

[18] R. Jhawar, V. Piuri, and M. Santambrogio, “Fault tolerance management in cloud computing: A system-level perspective,” *IEEE Systems Journal*, vol.7, no.2, pp.288 -297, 2012.

[19] W. Zhao, P. Melliar-Smith, and L. Moser, “Fault tolerance middleware for cloud computing,” *Proc. IEEE 3rd Int'l Conf. Cloud Computing (CLOUD '10)*, pp. 67–74, Jul. 2010.

[20] B. Guenter, N. Jain, and C. Williams, “Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning,”*Proc. 30th IEEE Int'l Conf. Computer Communications (INFOCOM '11)*, pp. 1332-1340, Apr. 2011.

[21] N. El-Sayed, I.A. Stefanovici, G. Amvrosiadis, A.A. Hwang, and B. Schroeder, “Temperature management in data centers: Why some (might) like it hot,” *ACM SIGMETRICS Performance Evaluation Review*, vol.40, no.1, pp.163-174, 2012.

[22] J.W. Lin, C.H. Chen ; J. M. Chang, “QoS-Aware Data Replication for Data-Intensive Applications in Cloud Computing Systems”, *IEEE Transactions on Cloud Computing*, vol.1,no.1,pp 101-115, 2013.

[23] K. Bilal, S. U. Khan, E. Calle, “On the Characterization of the Structural Robustness of Data Center Networks”, *IEEE Transactions on Cloud Computing*, vol.1, no.1, pp 64-77, 2013.

[24] R. Ghosh, F. Longo, F. Frattini, S. Russo, and K.S. Trivedi, “Scalable Analytics for IaaS Cloud Availability”, *IEEE Transactions on Cloud Computing*, vol.2, no.1, pp 57-70, 2013.

[25] K.V. Vishwanath, and N. Nagappan, “Characterizing cloud computing hardware reliability," *Proc. the 1st ACM Symp. Cloud computing (SOCC '10)*, pp. 193-204, Jun. 2010.

[26] Y. Zhang, M.S. Squillante, A. Sivasubramaniam, and R.K. Sahoo, “Performance implications of failures in large-scale cluster scheduling,” *Job Scheduling Strategies for Parallel Processing*, pp. 233-252, Springer, 2005.

[27] B. Schroeder, and G.A. Gibson, “A large-scale study of failures in high-performance computing systems,” *IEEE Transactions on Dependable and Secure Computing*, vol.7, no.4, pp. 337-350, 2010.

[28] R.K. Sahoo, M.S. Squillante, A. Sivasubramaniam, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," *Proc. Int'l Conf. Dependable Systems and Networks (DSN' 04)*, pp. 772-781, Jun. 2004.

[29] T. Guanhua, M. Dan, and Z. Jianfeng, "Reliable Resource Provision Policy for Cloud Computing," *Chinese Journal of Computers*, vol.10, no.33, pp.1859-1872, 2010.

[30] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, pp. 63-74, 2008.

[31] Cisco Data Center Infrastructure 2.5 Design Guide. http://www.cisco.com/univercd/cc/td/doc/solution/dcidg21.pdf.

[32] L. A. Barroso and U. H ¨olzle, "The Datacenter as a Computer:An Introduction to the Design of Warehouse-Scale Machines," *Synthesis Lectures on Computer Architecture* , vol. 4, no. 1,pp. 1–108, Jan. 2009.

[33] J. P. Srikanth and P. Bahl, "Flyways to De-congest Data Center Networks, *Proc. ACM SIGCOMM Workshop on Hot Topics in Networks (HotNet '09)*, Oct. 2009.

[34] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: a scalable fault-tolerant layer 2 data center network fabric," *Proc. ACM Int'l Conf. Data communication (SIGCOMM' 09)*, pp. 39-50, Aug. 2009.

[35] R. Sheldon. *A First Course in Probability*. Prentice Hall, 2002.

[36] M. Rausand, and A. Høyland. *System reliability theory: models, statistical approaches, and applications*. John Wiley & Sons., 2004

[37] H.C. Thomas, E.L. Charles, L.R. Ronald, and S. Clifford. *Introduction to algorithms*. MIT press, Cambridge, 2001.

[38] I. Goiri, F. Julia, and J. Guitart, "Efficient Data Management for Virtualized Service Providers," *Proc.17th Euromicro Conf. Parallel, Distributed and Network-based Processing*, pp. 409-413, Feb. 2009.

[39] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review* , vol. 37, no. 5, pp. 164–177, 2003.

[40] R. N. Calheiros, R. Ranjan, C. A. F. De Rose, and R. Buyya, "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services", Technical Report, Cloud Computing and Distributed Systems Laboratory, Department of Computer Science and Software Engineering,The University of Melbourne, 2009.

[41] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol.41, no.1, pp.23-50, 2011.

[42] A. Zhou, S. Wang, Q. Sun, H. Zou, and F. Yang, "FTCloudSim: a simulation tool for cloud service reliability enhancement mechanisms", *Proc. ACM/IFIP/USENIX International Middleware Conference, Demo&Poster Track*, Dec. 2013.

Ao Zhou received the M.E. degree in computer science and technology from the Institute of Network Technology, Beijing University of Posts and Telecommunications, in 2012. Currently, she is a Ph.D. candidate at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. Her research interests include cloud computing, service reliability.

Shangguang Wang is an associate professor at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He received his Ph.D. degree in computer science at Beijing University of Posts and Telecommunications of China in 2011. His PhD thesis was awarded as outstanding doctoral dissertation by BUPT in 2012. His research interests include Service Computing, Cloud Services, QoS Management. He is a member of IEEE.

Zibin Zheng is an associate research fellow at Shenzhen Research Institute, The Chinese University of Hong Kong. He received Outstanding Ph.D. Thesis Award of The Chinese University of Hong Kong at 2012, ACM SIGSOFT Distinguished Paper Award at ICSE2010, Best Student Paper Award at ICWS2010, and IBM Ph.D. Fellowship Award at 2010. His research interests include service computing and cloud computing

Ching-Hsien Hsu is a professor in department of computer science and information engineering at Chung Hua University, Taiwan. His research includes high performance computing, cloud computing, parallel and distributed systems, ubiquitous/pervasive computing and intelligence. He has been involved in more than 100 conferences and workshops as various chairs and more than 200 conferences/workshops as a program committee member. He is the editor-in-chief of international journal of Grid and High Performance Computing, and serving as editorial board for around 20 international journals.

Michael R. Lyu is currently a Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include software reliability engineering, distributed systems, service computing, information retrieval, social networks, and machine learning. He has published over 400 refereed journal and conference papers in these areas. Dr. Lyu is an IEEE Fellow and an AAAS Fellow for his contributions to software reliability engineering and software fault tolerance.

Fangchun Yang received his PhD degree in communication and electronic system from the Beijing University of Posts and Telecommunication in 1990. He is currently a professor at the Beijing University of Posts and Telecommunication, China. He has published 6 books and more than 80 papers. His current research interests include network intelligence, services computing, communications software, soft switching technology, and network security. He is a fellow of the IET.