BFTCloud: A Byzantine Fault Tolerance Framework for Voluntary-Resource Cloud Computing

Yilei Zhang[†], Zibin Zheng[†] and Michael R. Lyu[†§]
[†]*Department of Computer Science and Engineering*
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{*ylzhang, zbzheng, lyu*}*@cse.cuhk.edu.hk*
[§]*School of Computer Science*
National University of Defence Technology
Changsha, China

***Abstract*—Cloud computing is becoming a popular and important solution for building highly reliable applications on distributed resources. However, it is a critical challenge to guarantee the system reliability of applications especially in voluntary-resource cloud due to the highly dynamic environment. In this paper, we present BFTCloud (Byzantine Fault Tolerant Cloud), a Byzantine fault tolerance framework for building robust systems in voluntary-resource cloud environments. BFTCloud guarantees robustness of systems when up to f of totally $3f + 1$ resource providers are faulty, including crash faults, arbitrary behaviors faults, etc. BFTCloud is evaluated in a large-scale real-world experiment which consists of 257 voluntary-resource providers located in 26 countries. The experimental results shows that BFTCloud guarantees high reliability of systems built on the top of voluntary-resource cloud infrastructure and ensures good performance of these systems.**

I. Introduction

Cloud computing [1], [2] is Internet-based computing, whereby shared resources, software, and information are provided to computers and other devices on demand [3]. Currently, most of the clouds are deployed on two kinds of infrastructures. One is well-provisioned and well-managed infrastructure [4] managed by a large cloud provider (e.g., Amazon, Google, Microsoft, IBM, etc.). The other one is voluntary-resource infrastructure which consists of numerous user-contributed computing resources [5]. With the exponential growth of cloud computing as a solution for providing flexible computing resource, more and more cloud applications emerge in recent years. How to build high-reliable cloud applications, which are usually large-scale and very complex, becomes an urgent and crucial research problem.

Typically, cloud applications consist of a number of cloud modules. The reliability of cloud applications is greatly influenced by the reliability of cloud modules. Therefore, building high-reliable cloud modules becomes the premise of developing high-reliable cloud applications. Traditionally, testing schemes [6] are conducted on the software systems of cloud modules to make sure that the reliability threshold has been achieved before releasing the software. However, reliability of a cloud module not only relies on the system itself, but also heavily depends on the node it has deployed and the unpredictable Internet. Traditional testing has limited improvement on the reliability of a cloud module under voluntary-resource cloud infrastructure due to:

- Computing resources, denoted as nodes in the cloud, are frangible. Different from the powerful and performance-guaranteed nodes managed by large cloud providers, user-contributed nodes are usually highly dynamic, much cheaper, less powerful, and less reliable. The reliability of a cloud module deployed on these nodes is mainly determined by the robustness of nodes rather than the software implementation.
- Communication links between modules are not reliable. Unlike nodes in well-provisioned cloud infrastructure, which are connected by high speed cables, nodes in voluntary-resource cloud infrastructure are usually connected by unpredictable communication links. Communication faults, such as time out, will greatly influence the reliability of cloud applications.

Based on the above analysis, in order to build reliable cloud applications on the voluntary-resource cloud infrastructure, it is extremely urgent to design a fault tolerance mechanism for handling different faults. Typically, the reliability of cloud applications is effected by several types of faults, including: node faults like crashing, network faults like disconnection, Byzantine faults [7] like malicious behaviors (i.e., sending inconsistent response to a request [8]), etc. The user-contributed nodes, which are usually cheap and small, makes malicious behaviors increasingly common in voluntary-resource cloud infrastructure. However, traditional fault tolerance strategies cannot tolerate malicious behaviors of nodes.

To address this critical challenge, we propose a novel approach, called Byzantine Fault Tolerant Cloud (BFT-

978-0-7695-4460-1/11 $26.00 © 2011 IEEE
DOI 10.1109/CLOUD.2011.16

Cloud), for tolerating different types of failures in voluntary-resource clouds. BFTCloud uses replication techniques for overcoming failures since a broad pool of nodes are available in the cloud. Moreover, due to the different geographical locations, operating systems, network environments and software implementation among nodes, most of the failures happened in voluntary-resource cloud are independent of each other, which is the premise of Byzantine fault tolerance mechanism. BFTCloud can tolerate different types of failures including the malicious behaviors of nodes. By making up a BFT group of one primary and $3f$ replicas, BFTCloud can guarantee the robustness of systems when up to f nodes are faulty at run-time. The experimental results show that compared with other fault tolerance approaches, BFTCloud guarantees high reliability of systems built on the top of voluntary-resource cloud infrastructure and ensures good performance of these systems.

In summary, this paper makes the following contributions:

1) We identify the Byzantine fault tolerance problem in voluntary-resource cloud and propose a Byzantine fault tolerance framework, named BFTCloud, for guaranteeing the robustness of cloud application. BFTCloud uses dynamical replication techniques to tolerate various types of faults including Byzantine faults. We consider BFTCloud as the first Byzantine Fault Tolerant Framework in cloud computing literature.
2) We have implemented the BFTCloud system and test it on a voluntary-resource cloud, Planet-lab [1], which consists of 257 user-contributed computing resources distributed in 26 countries. The prototype implementation indicates that BFTCloud can be easily integrated into cloud nodes as a middleware.
3) We conduct large-scale real-world experiments to study the performance of BFTCloud on reliability improvement compared with other approaches. The experimental results show the effectiveness of BFTCloud on tolerating various types of faults in cloud.

The rest of this paper is organized as follows: Section II describes the system architecture of BFTCloud. Section III presents our BFTCloud fault tolerate mechanism in detail. Section IV introduces the experimental results. Section V discusses related work and Section VI concludes the paper.

II. System Architecture

We begin by using a motivating example to show the research problem in this paper. As shown in Figure 1, cloud applications usually consist of a number of modules. These modules are deployed on distributed cloud nodes and connected with each other through communication links. Each module is supposed to finish a certain task (e.g., product selection, bill payment, shipping addresses confirming, etc.)

[1]http://www.planet-lab.org

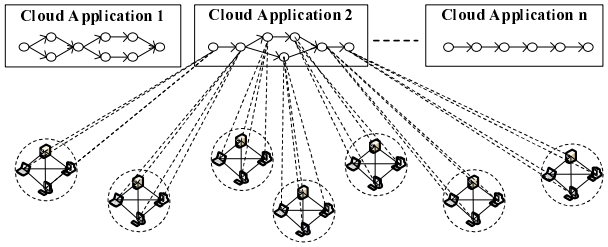


Figure 1. Architecture of Cloud Applications

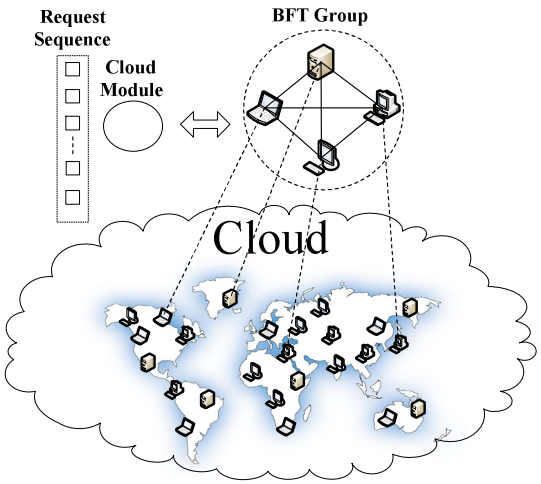


Figure 2. Architecture of BFTCloud in Voluntary-Resource Cloud

for a cloud application (e.g., shopping agency, etc.). A cloud module will form a sequence of requests (e.g., browsing products, choosing products, etc.) for the task (e.g, product selection, etc.) and send the requests to a group of nodes in the voluntary-resource cloud for execution.

Figure 2 shows the system architecture of BFTCloud in voluntary-resource cloud environment. Under the voluntary-resource cloud infrastructure, end-users contribute a larger number of computing resources which can be provided to cloud modules for request execution. Typically, computing resources in the voluntary-resource cloud are heterogeneous and less reliable, and malicious behaviors of resource providers cannot be prevented. Byzantine faults could be very common in a user-contributed cloud environment. In order to guarantee the robustness of the module, the replication technique is employed for request execution upon the user-contributed nodes. After a cloud module generated a sequence of requests, it first needs to choose a BFT group from the pool of cloud nodes for request execution. Since cloud nodes are located in different geographic locations with heterogeneous network environments, and the failure probabilities of nodes are diverse, a monitor is implemented on the cloud module side as a middleware for monitoring the QoS performance and failure probability of nodes. By considering the QoS performance and failure probability, the

cloud module first chooses a node as primary and send the current request to the primary. After that, a set of replicas are selected according to their failure probability and QoS performance to both the cloud module and the primary. The primary and replicas form a BFT group for executing requests from the cloud module. After the BFT group returns responses to the current request, the cloud module will judge whether the responses can be committed. Then the cloud module will send the next request or resend the current request to the BFT group. If some nodes of the BFT group are identified as faulty, the cloud module will update the BFT group to guarantee the system reliability. The detailed approach will be presented in Section III.

III. System Design

In this section, we present BFTCloud, a practical framework for building robust systems with Byzantine fault tolerance under voluntary-resource cloud infrastructure. We first give an overview on the work procedures of BFTCloud in Section III-A. Then we describe the five phases of BFTCloud in Section III-B to Section III-F respectively.

A. System Overview

Figure 3 shows the work procedures of BFTCloud. The input of BFTCloud is a sequence of requests with specified QoS requirements (e.g., preferences on price, capability, bandwidth, workload, response latency, failure probability, etc.) sent by the cloud module. The output of BFTCloud is a sequence of committed responses corresponding to the requests. BFTCloud consists of five phases described as follows:

1) **Primary Selection:** After accepting a request from the cloud module, a node is selected from the Cloud as the primary. The primary is selected by applying the primary selection algorithm with respect to the QoS requirements of the request.
2) **Replica Selection:** In this phase, a set of nodes are selected as replicas by applying a replica selection algorithm with respect to the QoS requirements of the request. The primary then forwards the request to all replicas for execution. The selected replicas together with the primary make up a BFT group.
3) **Request Execution:** In this phase, all members in the BFT group execute the request locally and send back their responses to the cloud module. After collecting responses from the BFT group within a certain period of time, the cloud module will judge the consistency of responses. If the BFT group respond consistently, the current request will be committed and the cloud module will send the next request. If the BFT group responds inconsistently, the cloud module will trigger a fault tolerance procedure to tolerate up to f faulty nodes and trigger the primary updating procedure and/or replica updating procedure to update the group members. If more than f nodes are identified as faulty, the cloud module will resend the request to the refresh BFT group and enter into the request execution phase again.
4) **Primary Updating:** In this phase, faulty primary in the BFT group will be identified and replaced by a newly selected primary.
5) **Replica Updating:** In this phase, faulty replicas in the BFT group will be identified and updated according to the information obtained from the request execution phase. The replica updating algorithm will be applied to replace the faulty replicas with other suitable nodes in the cloud.

B. Primary Selection

Under the voluntary-resource cloud infrastructure, a cloud module will send the request directly to a node which it believes to be the primary. Therefore, the primary plays a important role in a BFT group. The responsibilities of primary include: accepting requests from the cloud module, selecting appropriate replicas to form a BFT group, forwarding the request to all replicas, and replacing faulty replicas with newly selected nodes. Since failures happened on primary will greatly decrease the overall performance of a BFT group, the requirements on primary attributes (e.g., capability, bandwidth, workload, etc.) are more strict than those on replicas. In order to select an optimal primary, we propose a primary selection algorithm.

We model the primary selection problem under voluntary-resource cloud infrastructure as follows:

> *Let N be the set of nodes available in the cloud and Q be the set of m dimension vectors. For each node n_i in N, there is a $q_i = (q_{i1}, q_{i2}, \cdots, q_{im})$ in Q representing the QoS values of m criteria. Given a priority vector $W = (w_1, w_2, \cdots, w_m)$ on the m QoS criteria, the optimal primary should be select from the set N.*

Note that $w_k \in \mathbb{R}^+$ and $\sum_{k=1}^{m} w_k = 1$. Typically the QoS values of can be integers from a given range (e.g. $0, 1, 2, 3$ or real numbers of a close interval (e.g. $[-20, 20]$). Without loss of generality, we can map a QoS value to the interval $[0, 1]$ using the function $f(x) = (x - q_{min})/(q_{max} - q_{min})$, where q_{max} and q_{min} are the maximum and minimum QoS values of the corresponding criterion respectively.

The proposed primary selection algorithm is shown in Algorithm 1. After accepting the priority vector from the cloud module, a rating value r_i is computed for each node $n_i \in N$ as follows:

$$r_i = \sum_{k=1}^{m} q_{ik} \times w_k, \tag{1}$$

where r_i fall into the interval $[0, 1]$. The cloud module will choose the node n^*, which has the highest rating value, as

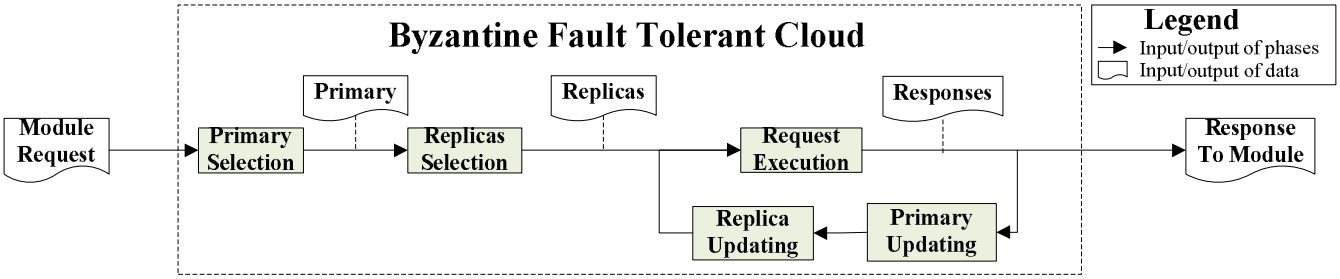


Figure 3. Work Procedures of BFTCloud

Algorithm 1: Primary Selection Algorithm

Input: N, Q, W
Output: n^*
$n^* = null$;
$r_{max} = 0$;
for *all* $n_i \in N$ **do**
  $r_i = \sum_{k=1}^{m} q_{ik} \times w_k$;
  if $r_i > r_{max}$ **then**
    $n^* = n_i$;
    $r_{max} = r_i$;
  end
end
return n^*;

the primary:

$$n^* = \arg\max_{n_i \in N} r_i. \tag{2}$$

C. Replica Selection

After the primary is selected in Section III-B, a set of replicas should be chosen to form a BFT group. Since replicas in a BFT group need to communicate with both the primary and the cloud module, the QoS performance of a node should be considered from both the cloud module perspective and the primary perspective. Let q_i be the QoS vector of node n_i observed by the cloud module and q'_i be the QoS vector of node n_i observed by the primary. Then the combined QoS vector q''_i is calculated by a set of transformation rules as follows:

- minimum: $q''_{ik} = min(q_{ik}, q'_{ik})$, for QoS criterion like bandwidth.
- average: $q''_{ik} = avg(q_{ik}, q'_{ik})$, for QoS criterion like response time.
- equality: $q''_{ik} = q_{ik} = q'_{ik}$, for QoS criterion like price.

Without loss of generality, the rule set can be easily extended to include more rules for calculating complex QoS criterion values.

Given the combined QoS vector q''_i, we can evaluate how appropriate the node n_i is as a replica of the BFT group. A score s_i is assigned to each node $n_i \in N$ as follows:

$$s_i = \sum_{k=1}^{m} q''_{ik} \times w_k, \tag{3}$$

where s_i falls into the interval $[0, 1]$. After ordering the scores, we can select the nodes ranked in high positions as replicas of the BFT group.

In order to decide the replication degree, we first consider the failure probability of a BFT group in its entirety. Since the BFTCloud guarantees the execution correctness when up to f nodes are faulty, a BFT group is faulty if and only if more than f nodes are faulty. We define the failure probability of a BFT group σ as follows:

$$P_\sigma \quad = P(|F| > f), \tag{4}$$

where F is the set of failure nodes in σ.

In order to reduce the cost of request execution, the replication degree f should be as small as possible, and the failure probability of a BFT group σ should be guaranteed under a certain threshold at the same time. Let P_0 be the threshold of P_σ defined by the cloud module. The replication degree decision problem can be formulated as an optimization problem:

$$\begin{aligned} \min_{f} \quad & f = \frac{|\sigma| - 1}{3}, \\ & P_\sigma = \sum_{F \in \Omega} \prod_{n_i \in F} P_i \prod_{n_j \in \sigma \setminus F} (1 - P_j), \\ & P_\sigma < P_0, \\ & \Omega = \{F | f < |F|\}. \end{aligned} \tag{5}$$

where P_i is the failure probability of node n_i, and Ω is the set of events that more than f nodes of the BFT group σ are fault. Note that a solution to this problem decides the replication degree and the replicas of BFT group σ at the same time. We summarize the replica selection algorithm in Algorithm 2.

D. Request Execution

After the BFT group members are determined, requests can be sent to the BFT group for execution. The cloud module first forms a request sequence and sends the sequence of requests to the primary. The primary will order the requests and forward the ordered requests to all the BFT group members. Each member of the BFT group will execute the

Algorithm 2: Replica Selection Algorithm

Input: N, Q, Q', W, P_0
Output: σ

$\sigma = null$;
$f = 0$;
$P_\sigma = P^*$;
for *all* $n_i \in N$ **do**
  $q_i'' \leftarrow (q_i, q_i')$ by applying the set of transformation rules;
  $s_i = \sum_{k=1}^{m} q_{ik}'' \times w_k$;
end
Generate a permutation $< n_1', n_2', \cdots >$ of the set N such that $s_1' \geq s_2' \geq \cdots$;
while $P_\sigma > P_0$ **do**
  $f = f + 1$;
  $\sigma = \{n_1', n_2', \cdots, n_{3f}'\}$;
  $P_\sigma = 0$;
  for *all* $F \in \Omega$ **do**
    $P_\sigma = P_\sigma + \prod_{n_i \in F} P_i \prod_{n_j \in \sigma \setminus F} (1 - P_j)$;
  end
end
return σ;

sequence of requests and send the corresponding responses back to the cloud module. The cloud module collects all the received responses from the BFT group members and make a judgement on the consistence of responses. A action strategy will be chose according to the consistence of responses as follows:

- **Case 1:** The cloud module receives $3f + 1$ consist responses from the BFT group. In this case, the cloud module will commit the current request since there is no fault happens in the current BFT group.
- **Case 2:** The cloud module receives between $2f + 1$ to $3f$ consist responses. In this case, the cloud module can still commit the current request since there are less than $f + 1$ faults happened. To commit the current request and identify the faulty nodes, the cloud module assembles a commit certificate and sends the commit certificate to all the BFT group members. Each member will acknowledge the cloud module with a local-commit message once it receives the commit certificate from the cloud module. If more than $2f + 1$ local-commit messages are received the cloud module will commit the current request and invokes a replica updating procedure to replace all the faulty BFT group members with new members. If less than $2f + 1$ local-commit messages are received, the cloud module will resend the commit certificate until it receives local-commit messages from more than $2f + 1$ members.
- **Case 3:** The cloud module receives less than $2f + 1$ response messages. In this case, either the primary is faulty or more than $f+1$ replicas are faulty. The cloud module will then resend the current request again but to all members this time. Each replica forwards the request to the node it believes to be the primary. If the replica receives a request from the primary within a given time and the proposed sequence number is consistency with that sent by the cloud module, the replica will execute the request and send response to the cloud module. If the replica does't receive an ordered request from the primary within a given time, or the request sequence number isn't consistency with the request sent by the cloud module, the primary must be faulty. The replica will send a primary election proposal to all replicas to trigger a primary updating procedure.
- **Case 4:** The cloud module receives more than $2f + 1$ responses, but fewer than $f + 1$ responses are consistency. This indicates inconsistent ordering of requests by the primary. The cloud module will send a proof of misbehavior of primary to all the replicas, and trigger a primary updating procedure.

E. Primary updating

When the primary is faulty, a primary updating procedures will be triggered in the Request Execution phase. The procedures of primary updating phase are as follows:

1) A replica which suspects the primary to be faulty sends an primary election proposal to all the other replicas. However, it still participates in the current BFT group as it may be only a network problem between the replica and the primary. Other replicas, once receiving a primary election proposal, just store it since the primary election proposal could arrive from a faulty replica as well.
2) If a replica receives $f + 1$ primary election proposals , it indicates that the primary is really faulty. It will send a primary selection request to the cloud module. The cloud module then will start the primary selection phase and return a new primary which is one of the current replicas. The replica then sends a primary updating message to all the other replicas, which includes the new primary name and $f + 1$ primary election proposals. Other replicas which receive such primary updating message again confirm that at least $f + 1$ replicas sent a primary election proposal, and then resend the primary updating message together with the proof to the new primary.
3) If the newly selected primary receives $2f + 1$ primary updating messages, it sends a new BFT group setup message to all the replicas, which again includes all the primary updating messages as proof.
4) A replica which received and confirmed the new BFT group setup message, will send out a BFT group confirm message to all replicas.
5) If a replica receives $2f + 1$ BFT group confirm messages, it starts performing as a memeber in the new BFT group.

F. Replica Updating

In the voluntary-resource cloud environment, nodes are highly dynamic and fragile. Different types of faults (e.g., response time out, unavailable, arbitrary behavior, etc.) may happen to the nodes after a period of time. Under voluntary-resource cloud infrastructure, the failure probability of a BFT group increases sharply as the fraction of faulty nodes increases. The failure probability of a BFT group under the condition that a set of replicas are already faulty is:

$$\begin{aligned} P_\sigma &= P(|F| > f | F^*) \\ &= P(|F \setminus F^*| > f - f^*), \end{aligned} \tag{6}$$

where F^* is the set of replicas which are faulty already.

To ensure the failure probability of a BFT group below a certain threshold, we need to replace the members once they are identified to be faulty. Moreover, due to the highly dynamic voluntary-resource cloud environment, the QoS performance of nodes are changed rapidly. Updating replicas timely could keep the performance of a BFT group stable.

Let T be the set of new nodes which will be added to the current BFT group. F^* is the set of nodes which will be removed from the current BFT group. Let σ' be the new BFT group with updated replicas. We have $\sigma' = \sigma \setminus F^* \cup T$, where T consists of nodes which are in the top $|T|$ positions ordered by score in Eq. (3).

The new BFT group σ', which can tolerate up to f' nodes failure, should satisfy $P_{\sigma'} > P_0$. Therefore, the replica updating problem is reduced to a replication degree decision problem, which can be further formalized as an optimization problem as follows:

$$\begin{aligned} \min_{f'} \quad & f' = \frac{|\sigma'| - 1}{3}, \\ & P_{\sigma'} = \sum_{F' \in \Lambda} \prod_{n_i \in F'} P_i \prod_{n_j \in \sigma' \setminus F'} (1 - P_j), \\ & P_{\sigma'} < P_0, \\ & \Lambda = \{F' | f < |F'|\}. \end{aligned} \tag{7}$$

where Λ is the set of events that more than f' nodes of the BFT group σ' are fault. We summarize the replica updating algorithm in Algorithm 3.

Algorithm 3: Replica Updating Algorithm

Input: N, Q, Q', W, P_0, σ, F^*
Output: σ'

$\sigma' = \sigma \setminus F^*$;
$T = null$;
$f' = \lceil \frac{3f - |F^*|}{3} \rceil$;
$P'_\sigma = P^*$;
for *all* $n_i \in N \setminus \sigma$ **do**
  $q''_i \leftarrow (q_i, q'_i)$ by applying the set of transformation rules;
  $s_i = \sum_{k=1}^{m} q''_{ik} \times w_k$;
end
Generate a permutation $< n'_1, n'_2, \cdots >$ of the set $N \setminus \sigma$ such that $s'_1 \geq s'_2 \geq \cdots$;
$T = \{n'_1, n'_2, \cdots, n'_{3f' - |\sigma'|}\}$;
$\sigma' = \sigma' \cup T$;
while $P'_\sigma > P_0$ **do**
  $f' = f' + 1$;
  $T = \{n'_1, n'_2, \cdots, n'_{3f' - |\sigma'|}\}$;
  $\sigma' = \sigma' \cup T$;
  $P'_\sigma = 0$;
  for *all* $F \in \Lambda$ **do**
    $P'_\sigma = P'_\sigma + \prod_{n_i \in F'} P_i \prod_{n_j \in \sigma' \setminus F'} (1 - P_j)$;
  end
end
return σ';

IV. Experiments

In this section, in order to study the performance improvements of our proposed approach, we conduct several experiments to compare our BFTCloud with several other fault tolerance approaches.

In the following, Section IV-A describes the system implementation of BFTCloud and the experimental settings, and Section IV-B compares the performances of BFTCloud with some other fault tolerance methods.

A. Experimental Setup

We have implemented our BFTCloud approach by Java language and deployed it as a middleware in a voluntary-resource cloud environment. The cloud infrastructure is constructed by 257 distributed computers located in 26 countries from Planet-lab, which is a distributed test-bed consisting of hundreds of computers all over the world. Each computer, named as node in the cloud infrastructure, can participate several BFT groups as a primary or replica simultaneously.

Based on the voluntary-resource cloud infrastructure, we conduct large-scale experiments study the performance improvements of BFTCloud compared with other approaches. In our experiments, each node in the cloud is configured with a random malicious failure probability, which denotes the probability of arbitrary behavior happens in the node. Note that the failure probability of a node observed by other nodes is not necessarily equal to the malicious failure probability since other types of faults (e.g., node crashing, disconnection, etc.) may also occur. Each node keeps the QoS information of all the other nodes and updates the information periodically. For simplicity, we use response-time for QoS evaluation in this paper. Without loss of generality, our approach can be easily extended to include more QoS criteria. We also employed 100 computers from Planet-lab to perform as cloud modules.

B. Performance Comparison

In this section, we compare the performance of our approach BFTCloud with other fault tolerance approaches

Table I
AVERAGE SENDING TIMES PER REQUEST

Benchmark	BFTCloud	BFTRandom	Zyzzyva	NoFT
0/0 KB	1.3428	1.7096	2.9167	1.0725
4/0 KB	1.3035	1.7248	3.1002	1.1042
0/4 KB	1.3820	1.7340	3.2058	1.3055

Table II
CORRECT RATE OF COMMITTED REQUESTS

size	BFTCloud	BFTRandom	Zyzzyva	NoFT
0/0 KB	0.9855	0.9468	0.8726	0.2589
4/0 KB	0.9840	0.9259	0.8925	0.2107
0/4 KB	0.9794	0.9278	0.8621	0.2216

in the voluntary-resource cloud environment. We have implemented four approaches:

- NoFT: No fault tolerance strategy is employed for task execution in the voluntary-resource cloud.
- Zyzzyva: A state-of-the-art Byzantine Fault tolerance approach proposed in [9]. The cloud module sends requests to a fixed primary and a group of replicas. There is no mechanism designed for adopting the dynamic voluntary-resource cloud environment.
- BFTCloud: The Byzantine Fault tolerance framework proposed in this paper. The cloud module employs Algorithm 1-3 to mask faults and adopt the highly dynamic voluntary-resource environment.
- BFTRandom: The framework is the same with BFTCloud. However, this approach just randomly selects nodes in primary selection, replica selection, primary updating, and replica updating phases.

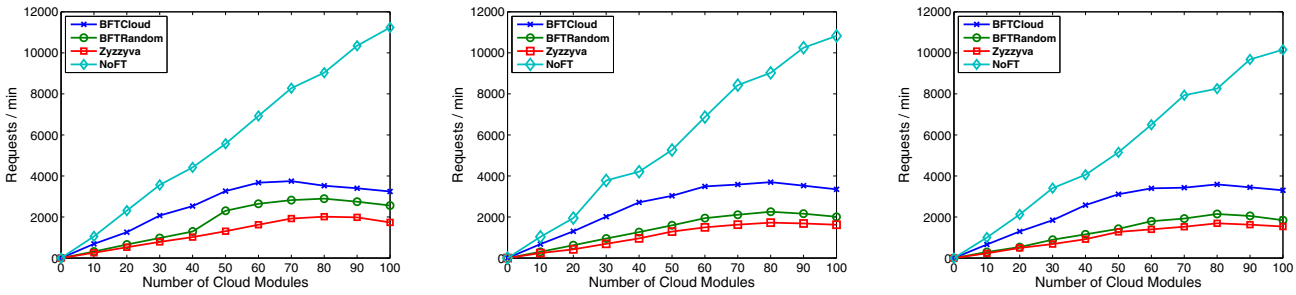
In Figure 4, we compare the throughput of all approaches in terms of different number of cloud modules by executing null operations. We change the number of cloud module from 0 to 100 with a step value of 10. The requests are sent by a variable number of cloud modules in each experiment (0-100). We conducts experiments on three benchmarks [9] with different request and response size. The sizes of request/response messages are 0/0KB, 4/0KB, and 0/4KB in Figure 4(a), Figure 4(b), and Figure 4(c) respectively. The parameter settings in this experiment are $P_0 = 0.5$ and $timeout = 500ms$, where $timeout$ defines the maximum waiting time for a message. From Figure 4, we can observe that our approach BFTCloud can commit more requests per minute than Zyzzyva and BFTRandom under different sizes of request/response messages. The reason is that BFTCloud always choose nodes with low failure probabilities as BFT group members. Therefore, the high reliability of BFT group guarantees that in most cases a request can be committed without be resent. Note that NoFT achieves the highest throughput among all approaches since NoFT employs no fault tolerance mechanism. Every request will be committed once the cloud module received a reply. However, NoFT cannot guarantee the correctness of committed requests, which will be discussed in Table II. Table I shows the average sending times of a request by the cloud module before it is committed. A request can be committed with much fewer sending times in BFTCloud than request in Zyzzyva, since BFT group members in BFTCloud are carefully selected and the probability of successfully executing a request is higher than that in Zyzzyva. Moreover, BFTCloud always choose nodes with good QoS performance as BFT group members which makes requests and responses are transmitted more quickly than other approaches. In general, BFTCloud achieves high throughput of committed requests which demonstrates that the idea of considering failure probability and QoS performance when selecting nodes is realistic and reasonable.

In Table II, we evaluate the correctness of committed requests of different approaches. The experimental result shows that among all the committed requests, the percentage of correctly committed requests is highest in BFTCloud. This is because BFTCloud can guarantee a low probability P_0 that more than f BFT group members are faulty. While Zyzzyva cannot guarantee the failure probability of BFT group since the primary and replicas in Zyzzyva are fixed. Most of the requests are not correctly committed in NoFT despite high throughput of NoFT, since no fault tolerance mechanism is employed.

V. RELATED WORK

Software fault tolerance techniques (e.g., N-Version Programing [10], distributed recovery block [11], etc.) are widely employed for building reliable systems [12]. Zhang et al. [13] propose a Web service search engine for recommending reliable Web service replicas. Salas et al. [14] propose an active strategy to tolerate faults in Web services. Zheng et al. [15] propose a ranking-based fault tolerance framework for building reliable applications in cloud. However, these techniques cannot tolerate Byzantine faults like malicious behaviors.

There are some works focus on Byzantine fault tolerance for Web services as well as distributed systems. BFT-WS [16] is a Byzantine fault tolerance framework for Web services. Based on Castro and Liskov's practical BFT algorithm [17], BFT-WS considers client-server application model running in an asynchronous distributed environment with Byzantine faults. $3f + 1$ replications are employed in the server-side to tolerate f Byzantine faults. Thema [18] is a Byzantine Fault Tolerant(BFT) middleware for Web services. Thema supports three-tiered application model, where the $3f + 1$ Web service replicas need to invoke an external Web service for accomplishing their executions. SWS [19] is a survivable Web Service framework that supports continuous operation in the presence of general failures and security attacks. SWS applies replication schemes



(a) Request/Response Size: 0/0 KB (b) Request/Response Size: 4/0 KB (c) Request/Response Size: 0/4 KB

Figure 4. Throughput Comparison for 0/0, 4/0, and 0/4 benchmarks as the number of cloud modules varies

and N-Modular Redundancy concept. Each web service is replicated into a service group to mask faults.

Different from above approaches, BFTCloud proposed in this paper aims to provide Byzantine fault tolerance for voluntary-resource cloud, in which Byzantine faults are very common. BFTCloud select voluntary nodes based on both their reliability and performance characteristics to adapt to the highly dynamic voluntary-resource cloud environment.

VI. Conclusion and Future Work

In this paper, we propose BFTCloud, a Byzantine fault tolerance framework for building reliable systems in voluntary-resource cloud infrastructure. In BFTCloud, replication techniques are employed for improving the system reliability of cloud applications. To adapt to the highly dynamic voluntary-resource cloud environment, BFTCloud select voluntary nodes based on their QoS characteristics and reliability performance. Faulty voluntary resources will be replaced with other suitable resources once they are identified. The extensive experimental results show the effectiveness of our approach BFTCloud on guaranteeing the system reliability in cloud environment.

In the future, we will conduct more experimental analysis on open-source cloud applications and conduct more investigations on different QoS properties of voluntary resources. We will conduct more experiments to study the impact of parameters and investigate the optimal values of parameters in different experimental settings.

Acknowledgment

The work described in this paper was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4154/10E) and sponsored in part by the National Basic Research Program of China (973) under Grant No. 2011CB302600.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] M. Creeger, "Cloud Computing: An Overview," *ACM Queue*, vol. 7, no. 5, pp. 1–5, 2009.

[3] T. Henzinger, A. Singh, V. Singh, T. Wies, and D. Zufferey, "FlexPRICE: Flexible Provisioning of Resources in a Cloud Environment," in *Proc. of CLOUD'10*, 2010, pp. 83–90.

[4] L. Tang, J. Dong, Y. Zhao, and L. Zhang, "Enterprise Cloud Service Architecture," in *Proc. of CLOUD'10*, 2010, pp. 27–34.

[5] A. Chandra and J. Weissman, "Nebulas: using distributed voluntary resources to build clouds," in *Proc. of HOTCLOUD'09*, 2009.

[6] M. Lyu, *Handbook of software reliability engineering*. McGraw-Hill, USA, 1996.

[7] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[8] Wikipedia, "http://en.wikipedia.org/wiki/byzantine_fault_tolerance."

[9] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative byzantine fault tolerance," in *Proc. of SOSP'07*, 2007, pp. 45–58.

[10] A. Avizienis, "The methodology of n-version programming," *Software fault tolerance*, pp. 23–46, 1995.

[11] K. Kim and H. Welch, "Distributed execution of recovery blocks: An approach for uniform treatment of hardware and software faults in real-time applications," *IEEE Transactions on Computers*, vol. 38, no. 5, pp. 626–636, 2002.

[12] M. Lyu, *Software fault tolerance*. John Wiley & Sons, USA, 1995.

[13] Y. Zhang, Z. Zheng, and M. Lyu, "WSExpress: A QoS-aware Search Engine for Web Services," in *Proc. of ICWS'10*, 2010, pp. 91–98.

[14] J. Salas, F. Perez-Sorrosal, M. Patiño-Martínez, and R. Jiménez-Peris, "WS-replication: a framework for highly available web services," in *Proc. of WWW'06*, 2006, pp. 357–366.

[15] Z. Zheng, T. Zhou, M. Lyu, and I. King, "FTCloud: A Component Ranking Framework for Fault-Tolerant Cloud Applications," in *Proc. of ISSRE'10*, 2010, pp. 398–407.

[16] W. Zhao, "BFT-WS: A Byzantine fault tolerance framework for web services," in *Proc. of EDOC'07*, 2008, pp. 89–96.

[17] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," *Operating Systems Review*, vol. 33, pp. 173–186, 1998.

[18] M. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan, "Thema: Byzantine-fault-tolerant middleware for web-service applications," in *Proc. of SRDS'05*, 2005, pp. 131–140.

[19] W. Li, J. He, Q. Ma, I. Yen, F. Bastani, and R. Paul, "A framework to support survivable web services," in *Proc. of IPDPS'05*, 2005, p. 93b.