Efficient Fault-Tolerant Strategy Selection Algorithm in Cloud Computing

P. Priyanka M.E. 1, A. Geetha. 2
1 PG Student, Department of Computer Science and Engineering, Nehru Institute of Technology, Coimbatore. Tamil nadu, India
2 Assistant professor, Department of Computer Science and Engineering, Nehru Institute of Technology, Coimbatore. Tamil nadu, India.

Abstract—Cloud computing is a mainstream feature of information technology. More progressively enterprises deploy their software systems in the cloud environment. The applications in cloud are usually large scale and containing a lot of distributed cloud components. Building cloud applications is highly reliable for challenging and critical research issues. Information processing systems has increased the significance of its correct and continuous operation even in the presence of faulty components. To address this issue, proposes a cloud framework to build fault-tolerant cloud applications. We first propose fault detection algorithms to identify significant components from the huge amount of cloud components. Then, we present an efficient fault-tolerance strategy selection algorithm to determine the most suitable fault-tolerance strategy for each significant component. Software fault tolerance is widely adopted to increase the overall system reliability in critical applications. System reliability can be enhanced by employing functionally equivalent components to tolerate component failures. Fault-tolerance strategies introduced a three well-known techniques are in the following with formulas for calculating the failure probabilities of the fault-tolerant modules. Our work will mainly be driven toward the implementation of the framework to measure the strength of fault tolerance service and to make an in-depth analysis of the cost benefits among all the stakeholders. An algorithm is proposed to automatically determine an efficient fault-tolerance strategy for the significant cloud components. Using real failure traces and model, we evaluate the proposed resource provisioning rules to determine their performance, cost as well as cost efficiency. The experimental results show that by tolerating faults of a small part of the most important components, the reliability of cloud applications can be highly improved.

Keywords—fault tolerance as a service, software reliability, Cloud application.

I. INTRODUCTION

Currently, the numeral of industrial cloud providers is quickly increasing. In an industrial multicloud atmosphere, individual providers are purposeful toward increasing their own revenue and do not care about the effectiveness of users and other providers. In such an atmosphere, we cannot trust the information obtained by the providers. As the influence of the Internet in solving large-scale problem is growing, the situation of having multiple self-interested agents is becoming more and more popular. In a industrial multicloud environment consisting of a set of selfish cloud providers with private information about their resources, application schedulers should act cautiously and not trust the information accepted by providers about the status of their resources, as there is always the opportunity of misrepresenting private information. For example, the providers might exaggerate the speed of their resources to increase their chance of earning more revenues.

Cloud computing is a new technology in academic world [1]. On cloud computing platform, resources are make available as service and by requirements, and it assurance to the subscribers that it sticks to the Service Level Agreement (SLA). On the other hand, because of the situation that the resources are shared, and the requirements of the subscribers have big dynamic heterogeneity and platform irrelevance, it will absolutely direct to resource waste if the resources cannot be distributed properly[2]. In addition, the cloud computing platform also requirements to enthusiastically balance the load among the servers in order to avoid hotspot and improve resource utility. Consequently, how to energetically and economically manage resources and to meet the requirements of subscribers become the problems to be solved.

Virtualization technology provides an effectual solution to the management of dynamic resources on cloud computing platform. Through sealing the service in virtual machines and mapping it to every physical server, the trouble of the heterogeneity and platform insignificance of subscribers’ requires can be better solved and at the same time the SLA is guaranteed. What is more, virtualization technology is capable to carry out remapping between virtual machine (VM) and physical resources according to the load change so as to achieve the scheduling of the whole system in a dynamic manner [3]. Consequently, virtualization technology is being methodically used in cloud computing. On the other hand, due to the highly energetic heterogeneity of resources on cloud computing platform, virtual machines must acclimatize to the cloud computing environment energetically so as to attain its best presentation by fully using its service and resources. But in order to progress resource utility, resources must be correctly allocated and load balancing must be guaranteed [4].

As a result, how to schedule VM resources to realize load balancing in cloud computing and to improve resource utility becomes an important research point.

Besides, with the development of cloud, the concept of "service" in conventional Service-Oriented Architecture (SOA) is unlimited from software application to comprehensive "cloud service" with the inclusion of both...
software applications and hardware equipments with good interoperability, self-organization and scalability [5]. The properties of cloud services have become more intricate and most of them necessitate superior computing ability to drive. While the benefits are enormous, this computing paradigm has considerably changed the dimension of risks on user’s applications, particularly because the failures (e.g., server overload, network congestion, hardware faults) that manifest in the data centers are outside the scope of the user’s organization [6], [7]. Nevertheless, these failures impose high implications on the applications deployed in virtual machines and, as a result, there is an increasing need to address users’ reliability and availability concerns.

The conventional way of achieving reliable and highly available software is to make use of fault tolerance methods at procurement and development time [8]. This implies that users must understand fault tolerance techniques and tailor their applications by considering environment precise parameters during the design phase. On the other hand, for the applications to be deployed in the Cloud computing environment, it is difficult to design a holistic fault tolerance solution that efficiently combines the failure behaviour and system architecture of the application. This difficulty arises due to: 1) high system complexity, and 2) abstraction layers of Cloud computing that release limited information about the underlying infrastructure to its users.

One of the most well-known software fault-tolerance techniques, also known as design diversity, is to occupy functionally equivalent yet independently designed components to tolerate faults [9]. Due to the cost of developing and maintaining redundant components, software fault tolerance is usually only employed for critical systems (e.g., airplane flight control systems, nuclear power station management systems, etc.). Different from traditional software systems, there are a lot of redundant resources in the cloud environment, making software fault tolerance a possible approach for building highly reliable cloud applications. Since cloud applications usually involve a large number of components, it is still too expensive to provide alternative components for all the cloud components. Moreover, there is probably no need to provide fault-tolerance mechanisms for the noncritical components, whose failures have limited impact on the systems. To reduce the cost so as to develop highly reliable cloud applications within a limited budget, a small set of critical components needs to be identified from the cloud applications. Our idea is also based on this well-known rule, i.e., by tolerating faults of a small part of the most important cloud components, the cloud application reliability can be greatly improved. Fault-tolerance strategies introduced a three well-known techniques are in the following with formulas for calculating the failure probabilities of the fault-tolerant modules. Our work will mainly be driven toward the implementation of the framework to measure the strength of fault tolerance service and to make an in-depth analysis of the cost benefits among all the stakeholders.

II. RELATED WORK

In [10] M. Castro et al presented a new replication algorithm which is able to tolerate Byzantine faults. In future Byzantine fault-tolerant algorithms are important because software errors and malicious attacks are increasingly common and can lead faulty nodes to display random behavior. While many algorithms alleged a synchronous system or were excessively slow to be used: These worked in asynchronous settings like the Internet and includes several significant optimizations which improve the response time of earlier algorithms by more than an order of magnitude. A Byzantine-fault-tolerant NFS service has been used performance has been measured. However the problem is large usage of resources since reducing the amount of resources required to implement our algorithm

In [11] B. Cully et al presented Whole-system replication is a familiar approach for providing high availability. Still, it typically has been used to be importantly more expensive than application-specific check pointing methods that only repeats relevant data. This approach perhaps used to bring HA “to the masses” as a platform service for VM’s. Despite the hardware and software constraints in which it operates and provides protection equal to or improved expensive commercial offerings. Several earlier methods actively used mirror persistent storage, requiring applications to do recovery from crash-consistent persistent state. On the contrary, Remus ensures that despite of the moment at which the primary fails, no superficially visible state is ever lost.

In [12] Y. Tamura et al proposes Kemari, a cluster system that synchronizes Virtual Machines for fault tolerance. It uses a possible approach to fault tolerance that does not need the use of exact hardware or alteration of applications. This method contains the design, implementation and evaluation of the system.

In [13] P. Narasimhan et al presented the Immune system which aims to offer survivability to CORBA applications and enabling them to continue to function in spite of malicious attacks, faults or accidents. Each and Every object inside the CORBA application is dynamically replicated by the Immune system, with majority voting applied on responses to each replica of the object and incoming invocations. The Secure Multicast Protocols are utilized to enable the majority voting to be useful, even when processors inside the network and objects surrounded by the application become corrupted.

In [14] N. Ayari et al presented Fault-tolerant frameworks which gives highly available services through fault detection and fault recovery methods. These frameworks require assembling different constraints based on the fault model performance, strength and consumption of resources. One of the factors that led to is the observation that present fault-tolerant frameworks are not constantly adapted to existing Internet services. Actually, many of the methods are not session-level-aware or transport-level, even though the concerned services varies from regular services like FTP and HTTP to more new Internet services such as voice over IP and multimodal conferencing. A comprehensive overview of fault tolerance concepts, approaches, and issues has been discussed in this work. It shows the redundancy of application servers
can be devoted to guarantee proficient failover of Internet services once the legitimate processing server goes down.

III. EXISTING SYSTEM

In Ravi Jhawar[15] present a theoretical framework, the Fault Tolerance Manager (FTM), that offers the origin for a service provider to recognize the delivery scheme and therefore to suggest fault tolerance as a service. Our goal is to insert our framework as a dedicated service layer between the client’s applications and the hardware that works openly on the top of the VM manager at the level of VM instances. The Fault Tolerance Manager have to address the subject of heterogeneity in computing resources, fulfill the goal of transparently providing fault tolerance support to user’s applications against node failures, and satisfy scalability and interoperability objectives. To deal with these challenges, [15] offer to build the Fault Tolerance Manager using the principles of service-oriented architecture, in which each ft−unit is appreciated as an individual web service, and a ft−sol is figured using the business process execution language (BPEL) constructs. It comprises a resource manager inside FTM that initially organizes with the cloud manager to generate the database and the resource graph. The resource manager is recognized in the form of a web service that presents a status operation that takes a resource (e.g., processing node, storage, memory) as input and outputs the state of that resource. Note that status operation can be run independently on each node and, using the update operation, state of the resource can be updated in the database and resource graph. For an example, Consider the start of the service invocation, the service provider produces a profile of computing resources in the cloud infrastructure by recognizing five processing nodes \( \{n_1, \ldots, n_5\} \in \mathbb{N} \) whose resource graph.

IV. PROPOSED WORK

Three famous fault-tolerance strategies are initiated in the following with formulas for estimating the failure probabilities of the fault-tolerant modules. In this work, failure probability of a cloud component is described as the probability that an invocation to this component will fail. The cost of failure probability is in the range of \([0,1]\).

Recovery Block (RB). Recovery block is a familiar mechanism utilized in software fault tolerance. It denotes structuring redundant program modules, where stand-in components will be appeal to sequentially if the primary component not passes. A recovery block fails simply if all the redundant mechanism fails. The failure probability \( f \) of a recovery block can be considered by:

\[
f = \prod_{i=1}^{n} f_i,
\]

where \( n \) is the number of redundant mechanism and \( f_i \) is the failure probability of the \( i \)-th part.

N-version programming (NVP). N-version programming, also known as multi version programming which is a software fault-tolerance method where multiple functionally equivalent programs also named as versions are autonomously generated from the same original specifications. While applying the NVP technique to the cloud applications, it independently implemented functionally equivalent cloud components are appeal to in parallel and the last result is decided by majority voting. The failure probability \( f \) of an NVP module can be calculated by

\[
f = \sum_{i=1}^{m} \frac{F(i)}{m}
\]

where \( n \) is the number of functionally equivalent mechanism \( (n \) is typically an odd number in NVP) and \( F(i) \) is probability that \( i \) choice components from all the \( n \) components fail.

Parallel. Parallel strategy appeal to all the \( n \) functional equivalent components in parallel and the first returned response will be utilized as the final result. A parallel module fails just if all the redundant components fail. The failure probability \( f \) of a parallel module can be calculated by

\[
f = \prod_{i=1}^{n} f_i
\]

where \( n \) is the number of redundant mechanism and \( f_i \) is the failure probability of the \( i \)-th part.

Efficient fault-tolerance strategy Selection

The fault-tolerance strategies have a number of distinctions based on different arrangements. For each major component that needs a fault-tolerance strategy, the designer can specify constraints. Two constraints are considered: one for response time and one for cost. The efficient fault-tolerance strategy selection problem for a cloud component with user constraints can then be formulated mathematically as

Minimize: \[
\sum_{i=1}^{m} f_i \times x_i
\]

\[
\sum_{i=1}^{m} s_i \times x_i \leq u_1
\]

\[
\sum_{i=1}^{m} t_i \times x_i \leq u_2
\]

\[
\sum_{i=1}^{m} x_i = 1
\]

\[
x_i \in \{0,1\}
\]

\( x_i \) is set to 1 if the \( i \)-th candidate is selected for the component and 0 otherwise. Furthermore, \( f_i, s_i, t_i \) and \( s_i \) are the failure probability, cost, and response time of the strategy candidates, respectively, \( m \) is the number of fault-tolerance strategy candidates for the component, and \( u_1 \) and \( u_2 \) are the user constraints for cost and response time, respectively.

We first calculate the cost, response time, and the aggregated failure probability values of different fault-tolerance strategy candidates employing the equations presented.
IV. EXPERIMENTAL RESULTS

![Graph showing the impact of component failure probability]

Fig.1 Impact of component failure probability

To analyze the impact of the component failure probability on the system reliability, we compare existing system under failure probability has to be setting between of 0.1 to 1 percent with a step value of 0.1 percent. With the increase of component failure probability from 0.1 to 1 percent, calculate the application failure probabilities of two approaches. From the results our proposed system has better than existing approach.

V. CONCLUSION AND FUTURE WORK

We presented method toward transparently delivering fault tolerance on the applications deployed in virtual machine instances. In specific, we presented a method for realizing generic fault tolerance approaches as independent modules, validating fault tolerance properties of each mechanism, and matching user’s requirements with available fault tolerance modules to obtain a comprehensive solution with desired properties. In our proposed component algorithms, the importance value of a component is decided by the number of components that invoke this component, the importance values of these components, how often the current component is invoked by other components, and the component fundamentals. After finding out the importance components, we propose an efficient fault-tolerance strategy selection algorithm to provide optimal fault-tolerance strategies to the importance components automatically, based on the constraints.

REFERENCES