ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

Improving The Performance For Failure-Aware Resource Provisioning In Cloud

Priyanka P[1], A Geetha[2], Biju Balakrishnan[3]

ME CSE Student, Nehru Institute of Technology, Tamilnadu, India[1]

Assistant Professor, CSE Dept., Nehru Institute of Technology, Tamilnadu, India[2, 3]

Abstract: Software fault tolerance is widely adopted to increase the overall system reliability in critical applications. System reliability can be improved by employing functionally equivalent components to tolerate component failures. The fault-tolerance strategies are calculating the failure probabilities of the fault-tolerant modules. The proposed policies take into account the workload model and the failure correlations to redirect users' requests to the appropriate Cloud providers. Using real failure traces and a workload model, we evaluate the proposed resource provisioning policies to demonstrate their performance, cost as well as performance–cost efficiency.

Key words: Cloud computing, fault tolerance as a service, fault tolerance properties, system level fault tolerance, fault tolerance strategies and performance-cost efficiency.

I. INTRODUCTION

The increasing demand for flexibility in obtaining and releasing computing resources in a cost-effective manner has resulted in a wide adoption of the Cloud computing paradigm. The availability of an extensible pool of resources for the user provides an effective alternative to deploy applications with high scalability and processing requirements spread across a geographical area. In general, a Cloud computing infrastructure is built by interconnecting large-scale virtualized data centers, and computing resources are delivered to the user over the Internet in the form of an on-demand service by using virtual Machines. While the benefits are immense, this computing paradigm has significantly changed the dimension of risks on user's applications, specifically 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. 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 traditional way of achieving reliable and highly available software is to make use of fault tolerance methods at procurement and development time. This implies that users must understand fault tolerance techniques and tailor their applications by considering environment specific parameters during the design phase. However, 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 behavior 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.

In contrast with the traditional approach, uses a new dimension where applications deployed in a Cloud computing infrastructure can obtain required fault tolerance properties from a third party. To support the new dimension, it extends the works and proposes an approach to realize general fault tolerance mechanisms as independent modules such that each module can transparently function on users' applications. This paper then enriches each module with a set of metadata that characterize its fault tolerance properties, and use the metadata to select mechanisms that satisfy user's requirements. Furthermore, presents a scheme that: 1) delivers a comprehensive fault tolerance solution to user's applications by combining selected fault tolerance mechanisms, and 2) ascertains the properties of a fault tolerance solution by means of runtime monitoring. Based on the proposed approach, this paper designs a framework that easily integrates with the existing Cloud infrastructure and facilitates a third party in offering fault tolerance as a service.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

II. SYSTEM OVERVIEW AND CONTRIBUTIONS

All components included in it are described in the following section. The described architecture is the modification of architecture in [1]. The additional functionalities added are the cost and performance evaluation. Hence the performance of the system improved. It also preserves the cost-performance efficiency.

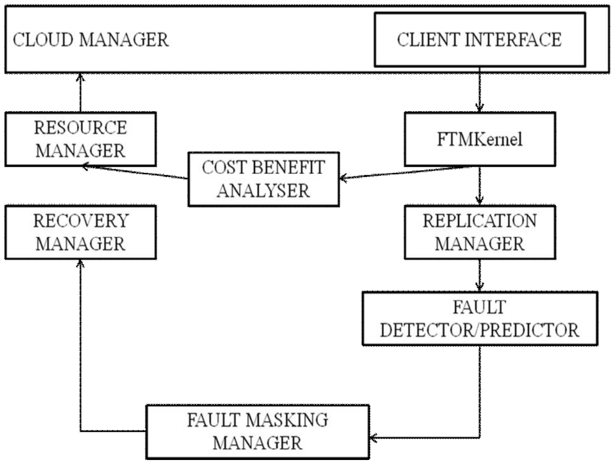

Fig 1: architecture of Fault Tolerance Manager

2.1CLIENT INTERFACE

The service invocation process begins when a client requests the service provider to offer fault tolerance support to its applications with a desired set of properties. In this context, it is essential to include a client interface component within FTM that provides a specification language that allows clients to specify and define their requirements. However, since the present-day cloud computing systems require its users to manage their VMs while dealing with sophisticated system-level concerns, an automated configuration tool that requires clients to simply select the application for which they wish to obtain fault tolerance support, and correspondingly provide values of desired availability, reliability, response time, criticality of the application and cost can be beneficial. This paper noted that an automated configuration tool can limit human errors and save time by lessening the need for manual tedious configuration. If the input can be provided in a high-level format(such as percentages, range, and numbers), even users with a nontechnical background can configure the desired properties with ease. Here, it considers the aspect of transforming high-level metric values into a set of fault tolerance properties, and translating the properties in terms of standard fault tolerance mechanisms.

2.2FTMKernel

The central computing component of our framework is the FTMKernel that is responsible for composing a fault tolerance solution based on client's requirements using the web service modules (ft–units) implemented by the service provider, delivering the composed service on client's applications, and monitoring each service instance to ensure its QoS. FTMKernel is composed of a service directory, a composition engine, and an evaluation unit.

1) Service directory: It is a registry of all ft−units realized by the service provider in the form of web services. A ft−unit applies a fault tolerance mechanism as a self-contained loosely coupled module, with a welldefined language-agnostic interface that: 1) describes its operations and input or output data structures (e.g., WSDL and WSCL), and 2) allows other ft−units to coordinate and assemble with it. In addition to the ft−units, this component also registers the metadata that represents the fault tolerance property p=(u,A) of each ft−unit. When FTM receives input from the client interface, this component first performs a matching between the client's preferences pc, and properties pi of each ft−unit in the service directory, to generate the set S of ft−units that satisfy pc. The set S is then ordered based
on client's preferences and provided to the composition engine.
2) Composition engine: It receives an ordered set of ft−units from the service directory as an input, and generates a comprehensive fault tolerance solution ft−sol using the web services (ft−units) that best match client's preferences as an output. In terms of service-oriented architecture, the composition engine can be viewed as a web service orchestration engine that exploits BPEL constructs to build a composed fault tolerance solution that is delivered to client's applications using robust message exchanges protocols.

3) Evaluation unit: It continuously monitors all composed
fault tolerance solutions at runtime using the validation function f (s,R) and the set of rules R defined corresponding to each ft−sol. We note that the interface exposed by web services (e.g., WSDL and WSCL) allows the evaluation unit to validate all the rules r∈R during runtime monitoring. If f (s,R) returns false, the evaluation unit updates the present attribute values in the metadata; otherwise, the service continues uninterrupted.

2.3RESOURCE MANAGER

The service provider must maintain a consistent view of all computing resources in the Cloud to efficiently allocate resources during each client request and to avoid over provisioning during failures. In this context, a resource manager that continuously monitors the working state of physical and virtual resources maintains a database of inventory and log information, and a graph representing the topology and working state of resources must be introduced by the service provider in the infrastructure provider's system.

The database of the resource manager must maintain the inventory information of each machine such as its unique serial number, composition of the machine (e.g., processor speed, number of hard disks, and memory modules), date when the machine was commissioned (or decommissioned), location of the machine in the cluster, and so on. The runtime state of machines such as memory used/free, disk capacity used/free, and processor cores utilization must also be logged. On the other hand, a resource graph must represent the topology of resources in a system.

2.4REPLICATION MANAGER

This component supports the replication mechanisms by invoking replicas and managing their execution based on the client's requirements. This paper denotes the set of VM instances that are controlled by a single implementation of a replication mechanism (ft−unit) as a replica group. Each replica within a group can be uniquely identified, and a set of rules R that must be satisfied by a replica group are specified. The task of the replication manager is to make the client perceive a replica group as a single service, and to ensure that the fault free replicas exhibit correct behavior during execution time.

2.5 FAULT DETECTION OR PREDICTION MANAGER

This component enriches the FTM by providing failure detection support at two different levels. The first level is infrastructure-centric, and provides failure detection globally across all the nodes in the Cloud, whereas the second level is application-centric and provides support only to detect failures among individual replicas within a replica group. To realize failure detection at either levels, this paper noticed that this component must support several well-known failure detection algorithms (e.g., the gossip based protocol, and heartbeat protocol) that are configured at runtime based on replication mechanisms and client's requirements.

2.6 FAULT MASKING MANAGER

The goal of this component is to support ft−units that realize fault masking mechanisms so that occurrence of faults in the system can be hidden from clients. When a failure is detected in the system, this component immediately applies masking procedures to prevent faults from resulting into errors. We note that the functionality of this component is critical to meet client's high availability requirements.

2.7 RECOVERY MANAGER

The goal of this component is to achieve system-level resilience by minimizing the downtime of the system during failures. To this aim, this component supports ft−units that realize recovery mechanisms so that an error-prone node can be resumed back to a normal operational mode. In other words, this component provides support that is complementary to that of the failure detection or prediction manager and fault masking manager, especially in the condition when an error is detected in the system. The FTM maximizes the lifetime of the Cloud infrastructure by continuously checking for occurrence of faults using the failure detection or prediction manager and, when exceptions happen, by recovering from failures using the recovery manager.

2.8COST BENEFIT FOR FAULT TOLERANCE

For each significant component that requires a fault tolerance strategy, the designer can specify constraints (e.g., response time of the component has to be smaller than 1,000 milliseconds, etc.). Two user constraints are considered: one for response time and one for cost. The optimal fault-tolerance strategy selection problems for a cloud component with user constraints are solved.

In Problem 1, x_i is set to 1 if the ith candidate is selected for the service and 0 otherwise. Selected for service means, there is no error in that candidate. Moreover, f_i, s_i, and t_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 service, and u_1 and u_2 are the user constraints for cost and response time, respectively.

To solve Problem 1, we first calculate the cost, response time, and the aggregated failure probability values of different fault-tolerance strategy candidates employing the equations presented. Then, Algorithm 1 is designed to select the optimal candidate. First, the candidates which cannot meet the user constraints are excluded. After that, the fault-tolerance candidate with the best failure probability performance will be selected as the optimal strategy for service i. By the above approach, the optimal fault-tolerance strategy, which has the best failure probability performance and meets all the user constraints, can be identified.

Problem 1. *Minimize:* $\sum_{i=1}^{m} f_i \times x_i$

Subject to:

- $\sum_{i=1}^{m} s_i \times x_i \leq u_1,$
- $\sum_{i=1}^{m} t_i \times x_i \leq u_2,$
- $\sum_{i=1}^{m} x_i = 1,$
- $x_i \in \{0, 1\}.$

Fig 2: minimization equation

2.8.1 Recovery blocks (RB):

Recovery block is a well known mechanism employed in software fault tolerance. A recovery block is a means of structuring redundant program modules, where standby components will be invoked sequentially if the primary component fails. A recovery block fails only if all the redundant components fail. The failure probability f of a recovery block can be calculated by:

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

where n is the number of redundant components and f_i is the failure probability of the ith component.

2.8.2 N-version programming (NVP):

N-version programming, also known as multiversion programming, is a software fault tolerance method where multiple functionally equivalent programs (named as versions) are independently generated from the same initial specifications. When applying the NVP approach to the cloud applications, the independently implemented functionally equivalent cloud service components are invoked in parallel and the final result is determined by majority voting. The failure probability f of an NVP module can be computed by

$$f = \sum_{i=\frac{n+1}{2}}^{n} F(i),$$

where n is the number of functionally equivalent components (n is usually an odd number in NVP) and F(i) is the probability that i alternative services from all the n services fail.

2.8.3 Parallel:

Parallel strategy invokes all the n functional equivalent components of a service in parallel and the first returned response will be employed as the final result. A parallel module fails only if all the redundant components fail. The failure probability f of a parallel module can be computed by

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

where n is the number of redundant components and f_i is the failure probability of the ith component.

These are the three mechanisms implemented to analyze cost of the fault tolerance.

III. CONCLUSION AND FUTURE WORK

In particular, this presented an approach for realizing generic fault tolerance mechanisms 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. After finding out the significant components, we propose an optimal fault-tolerance strategy selection algorithm to provide optimal fault-tolerance strategies to the significant components automatically, based on the user constraints. This developed a flexible and scalable Cloud architecture to solve the problem of resource provisioning for users' requests.

REFERENCES

[1] R. Jhawar, V. Piuri, and M. D. Santambrogio, "A comprehensive conceptual system-level approach to fault tolerance in cloud computing," in Proc. IEEE Int. Syst. Conf., Mar. 2012, pp. 1–5.
[2] W. Zhao, P. M. Melliar-Smith, and L. E. Moser, "Fault tolerance middleware for cloud computing," in Proc. 3rd Int. Conf. Cloud Comput., Jul. 2010, pp. 67–74.
[3] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in Proc. 3rd Symp. Operating Syst. Design Implementation, 1999, pp. 173–186.
[4] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: High availability via asynchronous virtual machine replication," in Proc. 5th USENIX Symp. Networked Syst. Design Implementation, 2008, pp. 161–174.
[5] Y. Tamura, K. Sato, S. Kihara, and S. Moriai, "Kemari: Virtual machine synchronization for fault tolerance," in Proc. USENIX Annu. Tech. Conf. (Poster Session), 2008.
[6] G. Koslovski, W.-L. Yeow, C. Westphal, T. T. Huu, J. Montagnat, and P. Vicat-Blanc, "Reliability support in virtual infrastructures," in Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci., Nov. 2010, pp. 49–58.
[7] P. Narasimhan, K. Kihlstrom, L. Moser, and P. Melliar-Smith, "Providing support for survivable CORBA applications with the immune system," in Proc. 19th IEEE Int. Conf. Distributed Comput. Syst., May 1999, pp. 507–516.
[8] N. Ayari, D. Barbaron, L. Lefevre, and P. Primet, "Fault tolerance for highly available internet services: Concepts, approaches, and issues," IEEE Commun. Surveys Tutorials, vol. 10, no. 2, pp. 34–46, Apr.–Jun. 2008.
[9] R. Guerraoui and M. Yabandeh, "Independent faults in the cloud," in Proc. 4th Int. Workshop Large Scale Distributed Syst, Middleware, no. 6. 2010, pp. 12–17.
[10] S. S. Kulkarni and K. N. Biyani and U. Arumugam, "Composing distributed fault-tolerance components," in Proc. Int. Conf. Dependable Syst. Netw., Supplemental Volume, Workshop Principles Dependable Syst., Jun. 2003, pp. W127–W136.
[11] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in Proc. 1st ACM Symp. Cloud Comput., 2010, pp. 193–204.
[12] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design. Englewood Cliffs, NJ: Prentice-Hall PTR, 2005.
[13] M. Hiltunen and R. Schlichting, "An approach to constructing modular fault-tolerant protocols," in Proc. 12th Symp. Reliable Distributed Syst., 1993, pp. 105–114.
[14] G. Vallee, K. Charoenpornwattana, C. Engelmann, A. Tikotekar, C. Leangsuksun, T. Naughton, and S. L. Scott, "A framework for proactive fault tolerance," in Proc. 3rd Int. Conf. Availability Reliability Security, Mar. 2008, pp. 659–664.
[15] W. Zhao, "Bft-ws: A Byzantine fault tolerance framework for web services," in Proc. 11th Int. Enterprise Distributed Object Comput. Conf. Workshop, Oct. 2007, pp. 89–96.
[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing," EECS Dept., Univ. California, Berkeley, UCB/EECS-2009-28, Feb. 2009.
[17] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati, "Encryption-based policy enforcement for cloud storage," in Proc. 30th Int. Conf. Distributed Comput. Syst. Workshop, 2010, pp. 42–51.
[18] P. Samarati and S. De Capitani di Vimercati, "Data protection in outsourcing scenarios: Issues and directions," in Proc. 5th ACM Symp. Inform. Comput. Commun. Security, 2010, pp. 1–14.
[19] F. Distante and V. Piuri, "Hill-climbing heuristics for optimal hardware dimensioning and software allocation in fault-tolerant distributed systems,"IEEE Trans. Reliab., vol. 38, no. 1, pp. 28–39, Apr. 1989.
[20] V. Piuri, "Design of fault-tolerant distributed control systems," IEEE Trans. Instrum. Meas., vol. 43, no. 2, pp. 257–264, 1994.
[21] L. L. Pullum, Software Fault Tolerance Techniques and Implementation.Norwood, MA: Artech House, 2001.