DOI 10.1109/SysCon.2012.6189503

A Comprehensive Conceptual System-Level Approach to Fault Tolerance in Cloud Computing

Ravi Jhawar*, Vincenzo Piuri*, Marco Santambrogio†
*Università degli Studi di Milano, Department of Information Technology, Crema, Italy
†Politecnico di Milano, Department of Electronics and Information, Milan, Italy
{ravi.jhawar, vincenzo.piuri}@unimi.it, marco.santambrogio@polimi.it

***Abstract*—Fault tolerance, reliability and resilience in Cloud Computing are of paramount importance to ensure continuous operation and correct results, even in the presence of a given maximum amount of faulty components. Most existing research and implementations focus on architecture-specific solutions to introduce fault tolerance. This implies that users must tailor their applications by taking into account environment-specific fault tolerant features. Such a need results in non transparent and inflexible Cloud environments, requiring too much effort to developers and users. This paper introduces an innovative perspective on creating and managing fault tolerance that shades the implementation details of the reliability techniques from the users by means of a dedicated service layer. This allows users to specify and apply the desired level of fault tolerance without requiring any knowledge about its implementation.**

I. Introduction

Cloud computing is gaining an increasing popularity over classic information processing systems for storing and processing huge volumes of data. This computing paradigm is built on modern data centers comprising thousands of interconnected servers with capability of hosting a large number of applications [21]. Most often, these data centers are virtualized, and computing resources are provisioned to the user as an on-demand services over the Internet in the form of configurable Virtual Machines (VMs) [1]. This dimension promises several benefits such as high scalability, mobility, lower entry and usage costs, and a semblance of infinite resources availability to the individual user.

On the other hand, the reliability of Cloud computing still remains a major concern among users. Due to economic pressures, these computing infrastructures often use commodity components exposing the hardware to scale and conditions for which it was not originally designed [21]. As a result, significantly large number of failures manifest in the system and seemingly impose high implications on the hosted applications, impacting their availability and performance. For example, Amazon's Elastic Compute Cloud (EC2) experienced failure in Elastic Block Storage (EBS) drives and network configuration, bringing down thousands of hosted applications and websites for 24-72 hours [4].

In this context, applications require fault tolerance abilities so that they can overcome the impact of system failures and perform their functions correctly when failures happen. At present, most Cloud delivery models require the application developers to build intrinsically reliable software taking into account environment-specific features [7]. However, this approach has certain limitations. Firstly, it demands knowledge expertise and experience from users in terms of selection, configuration, and integration of applications with available fault tolerance frameworks. Secondly, the granularity at which applications can handle failures, and resource costs incurred by the applications throughout the fail free period remains constant. Lastly, abstraction layers of the Cloud architecture result in non transparent and inflexible environments requiring too much effort to developers as little information about the underlying infrastructure is available.

Therefore, it is necessary to gain an impetus in the management of fault tolerance properties for the applications hosted in the Cloud computing environment. To achieve this, we introduce a new perspective, where applications can easily gain complementary fault tolerance properties from an external entity in the form of an on-demand service. The remainder of the paper is organized as follows. First, we summarize the related work in Section II. Then, after outlining the benefits of the new perspective over existing fault tolerance methodologies, we provide a comprehensive description of an approach which effectuates our envisioned perspective in Section III. We also design a framework to analyze the feasibility of the proposed approach. The architectural details of the framework is provided in Section IV, supported by a simple example in Section V. Finally, we present our conclusions in Section VI.

II. Related Work

A number of fault tolerance frameworks that allow applications to intrinsically gain resilience against failures have been proposed. In [17] the authors use active replication techniques for web services, and in [22] the authors propose a technique to gain byzantine fault tolerance using virtualization technology. Techniques to build efficient and fault tolerant applications for Amazon's EC2 are provided in [7]. Another approach using fault tolerance middleware which follows a leader/follower replication approach to tolerate crash faults has been proposed in [23]. However, all these techniques have the limitations mentioned in Section I, and either tolerate only a specific kind of fault or provide a single method to resilience.

In [8], [18], the authors present a high-availability technique in which applications that are encapsulated in VMs can transparently survive hardware failures using checkpointing and live-migration of VMs. We build on this technique and

construct a system-level fault tolerance service for Cloud computing which enforces fault tolerance operations at the VM-level.

The framework we propose depends on the feasibility of the modular perspectives for implementing fault tolerance algorithms and the efficient composition of modules which are two aspects addressed by some proposals (e.g. [14], [19]). In [19] the authors use a modular approach in developing a proactive fault tolerance framework to tailor a requirement-specific strategy, and demonstrate the framework to be a good platform for the study of various proactive fault tolerance policies. In [14] the authors propose an approach to build fault tolerance protocols by composing micro-protocols and combining them to a system using hierarchical techniques. This approach presents benefits in terms of easy *customization* when compared to a monolithic system and easy *experimentation* with different algorithms.

III. Our Approach

A. Rational Advantages of the new Perspective

Our proposal aims at overcoming the limitations of existing methodologies by offering fault tolerance properties to the applications as an on-demand service. The rational advantages of our approach are as follows.

- It provides flexibility to the applications to dynamically adjust its fault tolerance properties and the level of reliability and availability overtime. The resource costs can be made limited, and performance levels can be modified from one point to another based on business requirements. Achieving these features with traditional mechanisms would be extremely difficult.
- It simplifies the job of application developers since the users are only expected to specify desired properties for each application and have it delivered transparently. This relaxes the requirement of having knowledge expertise and experience both in reliable application development and managing fault tolerance/failures during runtime.
- The system-level complexity is also abstracted by the service layer. This allows users to receive a specifically composed fault tolerance support for its applications without requiring an in-depth knowledge about the system level procedures.

B. Outline of our approach

We propose to insert a dedicated service layer between the computing infrastructure and the applications which can offer fault tolerance support to each application individually while abstracting the complexity of the underlying infrastructure. To facilitate a full-fledged support, the service layer must contain a range of reliability mechanisms and must be able to create a fault tolerance solution with desired properties on-the-fly that can be delivered to the application. To achieve this, we build on the idea that a fault tolerance solution can be seen as a combination of a set of distinct activities coordinated in a specific event-based logic. For example, each fault tolerance mechanism, such as replication of an application, detection and masking of failures, and recovery of the erroneous system, can be viewed as a distinct activity that are coordinated in a sequential logic to form a general fault tolerance solution. In this perspective, we can realize each individual activity as a stand-alone, configurable module, that provides a coherent solution to a recurrent system failure in a given context. In addition, every module is associated with a set of metadata that describe the module's functional, operational, and structural properties. These metadata can be analyzed during runtime and compared with the users requirements to select appropriate activities. The chosen modules are then composed in a predefined logic to form an aggregate fault tolerance solution which is imposed on the users application. We propose to use loosely bound independent activities instead of a readily composed solution for two main reasons. First, changes to the users preferences can be made effective by re-composing or re-configuring the involved modules. Second, an individual module can be reused over several service instances saving resource costs.

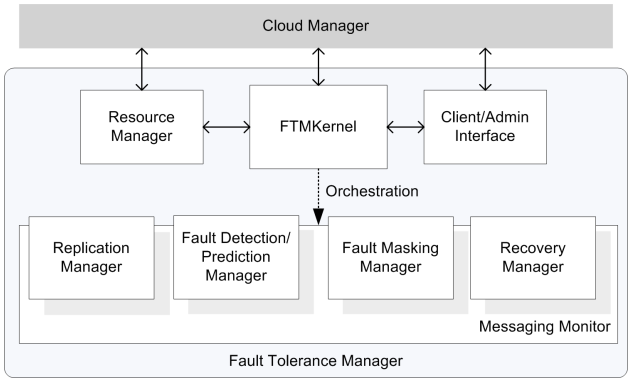

Fig. 1. FTM architecture

Our approach can be easily realized by implementing each module as an independent Web service because of their suitability to our application scenario. Web services expose a well-defined interface, in the form of a WSDL document [12], which can be used to implement the concept of metadata. The challenge of interoperability, portability, and integrability can be countered following the principles of Service-Oriented Architectures [12], and composite solutions can be dynamically created using BPEL-based orchestration engine [2], [6]. We study the feasibility of the proposed approach by designing a framework, the *Fault Tolerance Manager* (FTM). The architectural overview of FTM is presented in the next section.

IV. FTM Architecture: Overview

FTM is built to work on top of the hypervisor, spanning all the nodes and transversing the abstraction layers of the Cloud to transparently tolerate failures among the processing nodes. Fig. 1 illustrates the architecture of FTM which can primarily be viewed as an assemblage of several Web service components, each with a specific functionality. A brief description of the functionality of all the components along with the rationale behind their inclusion in the framework is

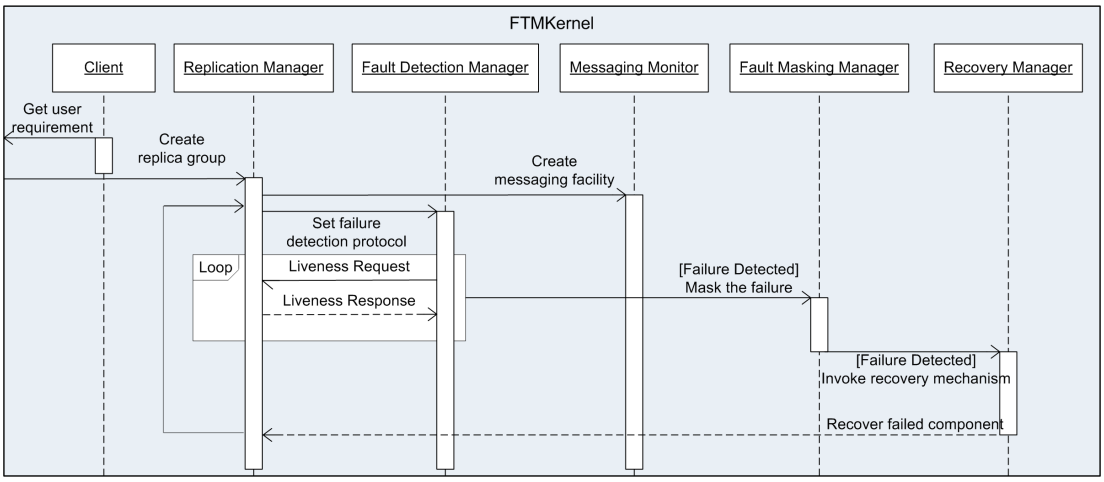

Fig. 2. A simple example of a general fault tolerance solution with composed work-flow orchestrated by FTMKernel.

provided further in this section.

Replication Manager. FTM provides fault tolerance by replicating users applications such that a redundant copy of the application is available after failure happens. In the Cloud computing environment, redundancy can also be applied on the entire VM instance in which the application is hosted. The set of VM instances controlled by a single implementation of the replication service is referred as a *Replica Group*. This component receives the reference of client's VM instance and expected replication properties such as the style of replication (active, passive, cold passive, hot passive) and the number of replicas in a replica group from the FTMKernel component. It then invokes the replica VM instances and ensures the rules defined for the group. Replication Manager also includes techniques to maintain consistency in a Replica Group by updating the state of backup replicas with that of the primary replica. Therefore, this component can be viewed as a collection of modules responsible for invoking the service replicas and managing their execution based on the user's requirements.

Fault Detection/Prediction Manager. To detect replica failures, well known algorithms such as Gossip-based protocol [20] and heartbeat message exchange protocol [23] are implemented following the approach presented in Section III and applied for this component. Every fault detection service must ideally detect the faults immediately after their occurrence and send a notification about the faulty replica to the FTMKernel to invoke services from the Fault Masking Manager and Recovery Manager. When a failure is detected, the Resource Manager is also notified to update the resource state of the Cloud.

Fault Masking Manager. FTM meets high availability demands by recovering or replacing the failed components in the background while the application's execution remains uninterrupted in another instance "replica". A collection of such algorithms that "mask" the occurrence of failures and prevent the faults from resulting into errors is included in this component. An example of a widely proposed and accepted masking technique in Cloud and virtualization environments is the Live Migration of VM instances [8], where the entire OS (VM instance) is moved to another location preserving the established session.

Recovery Manager. This component includes all the mechanisms that resumes error-prone nodes to a normal operational mode. The impact of failure detection and masking techniques on the system is complementary to that of recovery mechanisms. By continuously checking for the occurrence of faults and invoking the recovery service when exceptions happen, our framework maximizes the system's lifetime and minimizes the downtime during failures.

Messaging Monitor. FTM integrates WS-RM standard [3] with other Web service specifications like replication approach protocol so that the communication between any two components (and the replicas) is reliable even in the presence of component, system or network failure. This component extends through all the components of our framework (as shown in Fig. 1) and offers the necessary communication infrastructure in two different forms: message exchange among replicas of a replica group, and inter-component communication within the framework. FTMKernel chooses an appropriate messaging module while composing a fault tolerance algorithm such that an independent messaging facility is available for each instance.

Client/Admin Interface. This component is used to obtain users' requirements and act as an interface between the end user and FTM. We propose the use of an automated configuration tool which only requires the users to select the application or the VM instance to which they wish

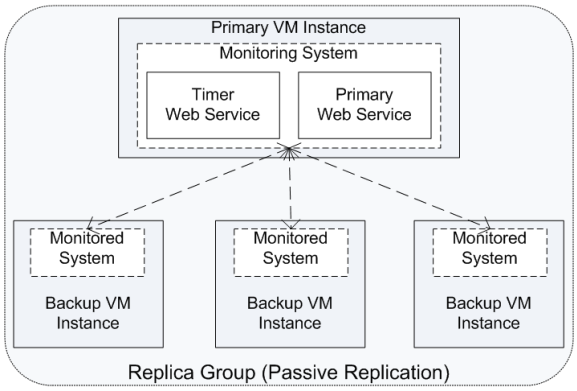

Fig. 3. Detecting VM instance crash failures using Heartbeat based protocol following the pull model

to obtain reliability support and correspondingly provide details of factors such as the desired availability, response time, criticality of the application, cost, and so on. In this manner, even users with a non-technical background can easily configure the system properties. After obtaining the users preferences, the Client interface forwards them to the FTMKernel.

FTMKernel. This is the central computing component of FTM which manages all the reliability mechanisms present in the framework. It contemplates the user's requirements and accordingly selects the Web (reliability) services from other components. The chosen modules are then orchestrated to form an aggregate solution that is delivered to the user's application. Fig. 2 shows an example workflow of activities between various chosen modules that forms a simple fault tolerance solution. In this example, FTMKernel first forms a replica group on receiving user's input. It then invokes the failure detection and messaging facilities for the service instance. If a failure is detected, the fault masking and recovery services are invoked.

Resource Manager. To achieve an efficient and proactive resource allocation, and avoid over provisioning during failures, the working state of the physical and virtual resources in the Cloud must be continuously monitored. The Resource Manager realizes this functionality in FTM, by maintaining a database including detailed logging information about the machines in the Cloud and providing an abstract, simple representation of the working state of resources in the form of a graph. This information can also be mined over time to derive the failure profile of the system and to design proactive fault tolerance models specially designated to the environment of the Infrastructure Provider.

V. Example: Heartbeat-based Protocol

This section provides a simple example to detect crash failures among VM instances within a Replica Group using our framework. To achieve this, FTMKernel invokes a multicast heartbeat based message exchange algorithm, following the pull model, as depicted in Fig. 3. In this algorithm, the *Primary*, which is responsible for contacting the *Backup Nodes*, activates the *Timer* every time it sends a liveness request to a node. The backup node immediately sends an acknowledgment to the primary upon receiving a request. The timeout period starts when the Timer is activated. When the timeout period ends, the Timer notifies the Primary. If N consecutive timeouts are overlapped for a particular VM instance, the node is suspected to have failed. In FTM, the heartbeat protocol can be realized by running a web service each for *Primary* and *Timer*, forming the Monitoring system, and the *BackupNode* service realizing the Monitored system. The timeout period can be set based on the users requirements. Since each service runs in the VM instance independent from the host application, no changes to its source code are required. This technique can be similarly extended to detect failures among nodes within various clusters in a Cloud.

VI. Conclusion

We introduced an innovative, system-level, modular perspective on creating and managing fault tolerance in Cloud computing environment. This high-level approach overcomes the inflexibility of application developers by shading the implementation details of the reliability techniques and offering the desired level of fault tolerance support as an on-demand service. The design of the framework which captures our new perspective, promises the feasibility of the envisioned approach to efficiently gain fault tolerance at system-level.

Our future work will be driven towards the implementation of the framework to realize the proposed approach and make an in-depth comparison of our perspective with existing monolithic systems. This study will also analyze the cost benefits of using our framework among all the stakeholders. Another aspect of our future work is to extend our approach in order to increase the reliability of the framework itself.

References

[1] Amazon elastic compute cloud. http://aws.amazon.com/ec2/.
[2] OASIS Web Services Business Process Execution Language Version 2.0 (WS-BPEL). http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.
[3] OASIS Web Services Reliable Messaging (WSRM). http://www.oasis-open.org/committees/tc-home.php?wg-abbrev=wsrm.
[4] Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region. http://aws.amazon.com/message/65648.
[5] "Security Guidance for Critical Areas of Focus in Cloud Computing," December 2009. [Online]. Available: https://cloudsecurityalliance.org/csaguide.pdf
[6] A. Albreshne, P. Fuhrer, and J. Pasquier, "Web Services Orchestration and Composition: Case Study of Web services composition," September 2009.
[7] J. Barr, A. Narin, and J. Varia, "Building Fault-Tolerant Applications on AWS," October 2011. [Online]. Available: http://media.amazonwebservices.com/AWS-Building-Fault-Tolerant-Applications.pdf
[8] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: high availability via asynchronous virtual machine replication," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 161–174.

[9] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati, "Encryption-Based Policy Enforcement for Cloud Storage," in *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems Workshops*, ser. ICDCSW '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 42–51.

[10] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Encryption policies for regulating access to outsourced data," *ACM Trans. Database Syst.*, vol. 35, pp. 12:1–12:46, May 2010.

[11] F. Distante and V. Piuri, "Hill-climbing heuristics for optimal hardware dimensioning and software allocation in fault-tolerant distributed systems," *Reliability, IEEE Transactions on*, vol. 38, no. 1, pp. 28 –39, apr 1989.

[12] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[13] A. Heddaya and A. Helal, "Reliability, Availability, Dependability and Performability: A User-centered View," Boston, MA, USA, Tech. Rep., 1997.

[14] M. Hiltunen and R. D. Schlichting, "An Approach to Constructing Modular Fault-Tolerant Protocols," in *In Proceedings of the 12th Symposium on Reliable Distributed Systems*. IEEE, 1993, pp. 105–114.

[15] V. Piuri, "Design of fault-tolerant distributed control systems," *Instrumentation and Measurement, IEEE Transactions on*, vol. 43, no. 2, pp. 257 –264, apr 1994.

[16] P. Samarati and S. De Capitani di Vimercati, "Data protection in outsourcing scenarios: issues and directions," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '10. New York, NY, USA: ACM, 2010, pp. 1–14.

[17] G. T. Santos, L. C. Lung, and C. Montez, "FTWeb: A Fault Tolerant Infrastructure for Web Services," in *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, ser. EDOC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 95–105.

[18] Y. Tamura, K. Sato, S. Kihara, and S. Moriai, "Kemari: Virtual Machine Synchronization for Fault Tolerance," in *Proceedings of USENIX Annual Technical Conference*, 2008.

[19] G. Vallee, K. Charoenpornwattana, C. Engelmann, A. Tikotekar, C. Leangsuksun, T. Naughton, and S. L. Scott, "A Framework for Proactive Fault Tolerance," in *Proceedings of the 2008 Third International Conference on Availability, Reliability and Security*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 659–664.

[20] R. van Renesse, Y. Minsky, and M. Hayden, "A gossip-style failure detection service," in *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, ser. Middleware '98. London, UK: Springer-Verlag, 1998, pp. 55–70.

[21] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 193–204.

[22] T. Wood, R. Singh, A. Venkataramani, P. Shenoy, and E. Cecchet, "ZZ and the art of practical BFT execution," in *Proceedings of the sixth conference on Computer systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 123–138.

[23] W. Zhao, P. M. Melliar-Smith, and L. E. Moser, "Fault Tolerance Middleware for Cloud Computing," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, ser. CLOUD '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 67–74.