Fault Tolerance Management in IaaS Clouds

Ravi Jhawar and Vincenzo Piuri
Dipartimento di Informatica – Università degli Studi di Milano, 26013 Crema, Italy
Email: *firstname.lastname*@unimi.it

***Abstract*—Fault tolerance, reliability and availability in Cloud computing are critical to ensure correct and continuous system operation also in the presence of failures. In this paper, we present an approach to evaluate fault tolerance mechanisms that use the virtualization technology to transparently increase the reliability and availability of applications deployed in the virtual machines in a Cloud. In contrast to several existing solutions that assume independent failures, we take into account the failure behavior of various server components, network and power distribution in a typical Cloud computing infrastructure, the correlation between individual failures, and the impact of each failure on user's applications. We use this evaluation to study fault tolerance mechanisms under different deployment contexts, and use it as the basis to develop a methodology for identifying and selecting mechanisms that match user's fault tolerance requirements.**

***Index Terms*—Fault Tolerance as a Service, Fault Tolerance Management, Infrastructure Clouds**

I. Introduction

Cloud computing is gaining an increasing popularity over traditional information processing systems. It offers immense benefits in terms of flexibility in obtaining and releasing computing resources, as and when required, in a cost-effective manner. As a consequence, this paradigm is widely being used particularly to deploy applications with high scalability, processing and storage requirements.

Due to economic limitations, Cloud infrastructures are often built using commodity components, and as a consequence, the hardware is exposed to scale and conditions it was not originally designed for [13]. Furthermore, due to very high system complexity, even carefully engineered data centers are subject to a large number of failures, especially when they are distributed in several locations. The dimension of risks on user's applications is significantly changed since failures in the data centers are outside the scope of user's organization. Traditional fault tolerance approaches are therefore less effective, and there is a pressing need to address user's reliability and availability concerns as one of the basis to improve the overall system's security.

The traditional way to increase reliability and availability of software is to employ robust fault avoidance and fault tolerance techniques at development time. In this approach, users must take into account the system architecture and build their applications accordingly. However, the system's architectural details are not widely available to the users because of the abstraction layers and business model of Cloud computing. As an alternative, a new perspective of offering fault tolerance as an additional service either by a third party or the service provider itself is being advocated [6], [7], [9]. This new approach leverages existing fault tolerance mechanisms that use virtualization technology and its capabilities to transparently replicate and migrate virtual machine (VM) instances. In this direction, we developed a conceptual framework, the Fault Tolerance Manager (FTM), which includes all the components necessary to realize the new perspective [6].

In this paper, we build on the principles of the FTM, and present a solution on two important aspects of the service that were not analyzed previously. First, we present an approach to measure the effectiveness of a fault tolerance mechanism built using the virtualization technology. To achieve this, we evaluate the level of reliability and availability that can be obtained by using a particular fault tolerance mechanism at different deployment levels using fault trees [12] and Markov models. Second, we present a methodology to select the fault tolerance mechanisms that most appropriately match user's requirements by considering the effectiveness measure (that is obtained in the first step). This matching process is essential for the FTM to correctly deliver the fault tolerance support.

The remainder of the paper is as follows. Section II describes the motivating scenario. Section III presents an overview of a typical Cloud infrastructure, outlines the failure behavior of critical system components and their impact on the service. Section IV discusses representative fault tolerance mechanisms that use virtualization techniques to transparently obtain fault tolerance and describes possible deployment scenarios. Section V presents an approach to identify and select fault tolerance techniques based on user's requirements. Section VI summarizes the related work and Section VII outlines our conclusions.

II. Motivating Scenario

In our study, we consider a service-oriented and distributed Cloud computing environment that involves the following stakeholders.

- *Infrastructure Provider* (IP): realizes a Cloud computing infrastructure, and delivers computing resources to its users as an on-demand service.
- *User* (U): deploys its applications using infrastructure provider's service. We assume that a user satisfies its reliability and availability requirements by leveraging the service offered by the fault tolerance service provider.
- *Fault Tolerance Service Provider* (SP): offers fault tolerance support to user's applications based on a given set of requirements. We assume that the service provider is trusted both by IP and U.

978-1-4673-4688-7/12/$31.00 ©2012 IEEE

As an example, consider a user offering a web-based banking service which allows its customers to manage their accounts over the Internet. The user implements the banking service as a multi-tier application that uses the storage service offered by the IP to store and retrieve its customer data, and compute service to process its operations and respond to customer queries. In this context, we note that a failure in the storage server or compute nodes may highly impact the banking service. Furthermore, each tier of the banking application may require different fault tolerance properties, and the requirements may change over time based on business demands. By using traditional approaches, fault tolerance of the banking service remains constant throughout its life-cycle. Hence, it is easier for the user to specify its requirements and make use of the service offered by the SP to meet its reliability and availability goals. However, in this context the service provider clearly requires the ability to identify the failure characteristics of the system, evaluate different configurations of various fault tolerance mechanisms it has implemented, and quantify the reliability and availability obtained by each mechanism to suitably deliver and maintain its service to the user. It also requires a mechanism that allows users to specify its requirements with ease, and a scheme to match user's high level requirements with its low level techniques.

III. Failure Characteristics

A Cloud computing user must engage with the service provider to obtain fault tolerance support for its applications. The goal of the service provider is to create a fault tolerance solution based on user's requirements and realize the solution by taking into account the failure characteristics of the Cloud infrastructure. In this section we first present an overview of a typical Cloud infrastructure and then derive an approach to characterize the failures in the system.

A. Overview of Cloud infrastructure

We consider that the Cloud infrastructure is developed by interconnecting large-scale data centers that host thousands of servers. Each server contains multiple processors, storage disks, memory modules and network interfaces. A hypervisor is deployed on each server to virtualize its resources, and required amounts of computing resources are delivered to the user in the form of virtual machine instances. All the servers are connected using several network switches and routers. In particular, as described in [4], we consider that servers are first connected via a 1Gbps link to a switch (S), which is in turn connected to two (primary and backup) aggregation switches (AggS). The subsystem formed by the group of servers under an aggregate switch can be viewed as a cluster. An AggS connects tens of switches (S) to redundant access routers (AccR). This implies that each AccR handles traffic from thousands of servers and route it to core routers that connect different data centers to the Internet.

B. Failure behavior of system components

Failure behavior of the system must be modeled in the service provider's perspective, that is, the infrastructure component failures which result in a user application failure must be implicitly represented. We model the failure behavior using the notion of fault trees [8], [12] since the dependence between individual failures in the Cloud infrastructure and the boundaries on the impact of each failure can be taken into account. In particular, we consider failures on three main types of resources: server, network and power.

- *Server*: An application deployed in a VM instance that is hosted on a server may fail if there is a failure in the physical host or management software. In other words, a failure/error either in the *i)* processor, memory modules, storage disks, power supply or network interfaces, or *ii)* the virtual machine manager (VMM), or *iii)* the VM instance itself, may lead the application to a failure. Figure 1 illustrates this behavior as a fault tree where the top-event represents the failure in user's application (i.e., when the top-event's value is `true`). Service provider can determine the reliability and availability of a server and its components using Markov models (see Section IV-A).
- *Network*: Figure 2 represents the fault tree for an application considering the network failures in the system, based on the network architecture described in Section III-A. A failure in this context implies that the application is not connected to the rest of the network or gives errors during data transmission. We note that the fault tree clearly defines the boundaries (using server, cluster and data center level blocks) and impact of network failures. This allows the service provider to increase the fault tolerance of user's applications (e.g., by placing individual replicas of an application in different failure zones). A network failure happens if there is an error in all redundant switches S, AggS, AccR or core routers, or the network links connecting the physical host and other network components.
- *Power*: We assume that a data center receives the power via an uninterrupted power network, and a redundant distribution unit (DU) is deployed for each cluster within the data center. A DU provides power to all the servers within a cluster. A failure in the DU is independent of other DUs and the central power supply. Figure 3 depicts the fault tree of power failures in a Cloud infrastructure.

We note that this method can be extended to incorporate other failures (e.g., cloud manager errors) in a straightforward manner, and are not included in this paper due to space constraints. We use the failure characteristics and fault trees to select an appropriate deployment configuration for the fault tolerance mechanism applied on an application (see Section IV-B).

IV. Fault tolerance in Cloud computing

We discuss about representative fault tolerance mechanisms that can transparently handle system component failures and possible deployment scenarios in Cloud infrastructures.

A. Fault Tolerance Mechanisms

The task of offering fault tolerance as a service requires the service provider to realize fault tolerance mechanisms that

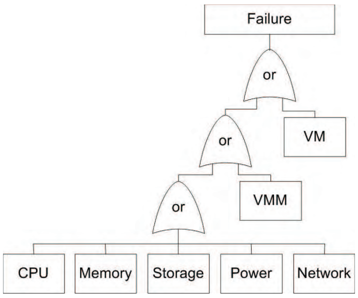

Fig. 1. Fault tree for server failure

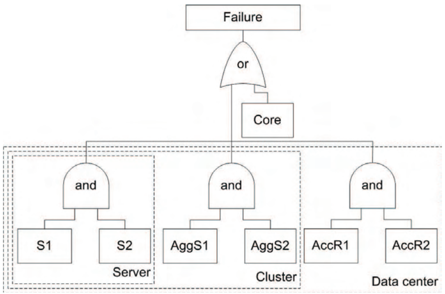

Fig. 2. Fault tree for network failure

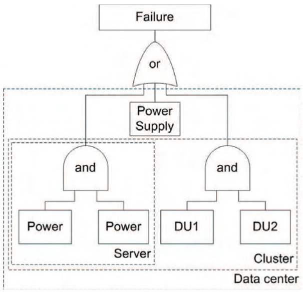

Fig. 3. Fault tree for power failure

can transparently function on user's applications. We define ft_sol as an independent module that applies a coherent fault tolerance mechanism to a recurrent set of system failures at the granularity of a VM instance. The notion of ft_sol is based on the observation that the impact of hardware failures on user's applications can be handled by applying fault tolerance mechanisms directly at the virtualization layer than the application itself. For instance, fault tolerance of the banking service can be increased by replicating the entire VM instance in which its application-tier is deployed on multiple physical nodes. We present here a brief discussion on three main configurations of ft_sols based on replication schemes, which represent the majority of fault tolerance implementations that are currently being used.

1) Semi-active replication: The input is either provided to all the replicas or state information of the primary replica is frequently transmitted to the backup replicas. The primary as well as the backup replicas executes all the instructions, but only the output generated by the primary replica is made available to the user. The output messages of backup replicas are logged by the hypervisor. In case the primary replica fails, one of the backup replicas can readily resume the service execution. For each replica failure, the FTM must create an equivalent replica (VM instance) on another host and update its state. We note that in a cloud computing environment, resources are often over-provisioned, and hence it is possible to create backup resources with a very high probability. An example of a technique that falls in this category is the VMware's Fault Tolerance [15] that is designed for mission-critical workloads. We note that the availability obtained by using this technique is very high, but it comes at high resource consumption costs.

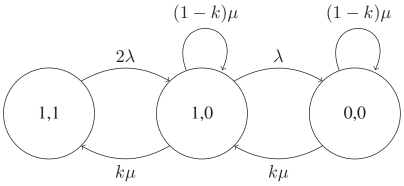

Fig. 4. An example of a Markov model for semi-active replication. λ is the failure rate and μ is the recovery rate

As discussed in the previous section, we use Markov models to determine the reliability and availability of the application that uses this replication scheme because failure behavior of physical hosts (servers) must be taken into account. Figure 4 depicts the Markov model of a representative ft_sol that is based on semi-active replication scheme with two replicas. Each state is represented as (x, y) where x=1 implies that the primary replica is working and x=0 implies that it failed. Similarly, y represents the state of the backup replica. The system starts and remains in state (1,1) during normal execution, i.e., when both replicas are available. When a VM instance (either primary or backup replica) fails, the system moves to state (0,1) or (1,0) where other replica takes over the execution process. We note that a single state is sufficient to represent this condition in the Markov model since, in the service provider's perspective, both the replicas are equivalent. In state (0,1) or (1,0), FTM initiates the recovery mechanism defined in the ft_sol, and the system moves to state (1,1) if the recovery is successful; if the server experiences a failure, the system transits to state (0,0) where the service becomes unavailable.

2) Semi-passive replication: The state information is obtained by frequently checkpointing the primary replica and buffering the input parameters between each checkpoint, and replication is performed by transferring the state information to the backup replicas. The backup replicas do not execute the instructions but saves the latest state obtained from the primary replica. In case the primary replica fails, a backup replica is initiated and updated to the current state with some loss in the present execution cycle and reasonable downtime. Remus [3] is a typical example of a system that is used by the Xen hypervisor and realizes a configuration of semi-passive replication. We note that the availability obtained from this technique is less than that by semi-active replication, but the resource consumption costs are reduced since the backup replicas do not execute instructions.

Figure 5 represents the Markov model of an application for which the semi-passive replication mechanism with two replicas is applied by the service provider. In this model, when a failure in the primary replica happens, the system moves from state (1,1) to state (0,1) and begins the update process. The backup replica assumes the execution process (becomes the new primary replica) and the system implicitly

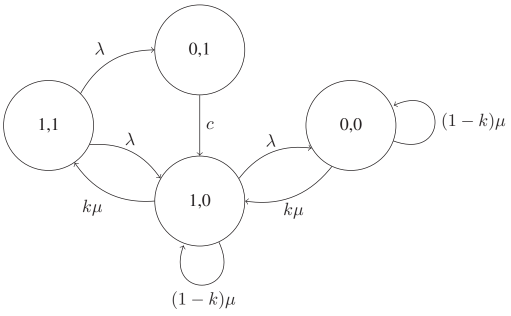

Fig. 5. An example of a Markov model for semi-passive replication

moves to state (1,0). In this state, FTM invokes a new replica and provides it with the latest checkpoint. If the new backup replica is successfully commissioned, the system again moves to state (1,1), otherwise it remains in state (1,0). A failure in the primary replica in state (1,0) results in a complete system failure (i.e., both replicas become unavailable), and the system transits to state (0,0).

3) Passive replication: The state information of a VM instance is regularly stored on a backup. In case of a failure, FTM recommissions another VM instance and restores the last saved state. We note that a backup can be configured to share the state of several VM instances or it can be dedicated to a particular application, and the VM recommissioning process can be performed based on a priority value assigned to each VM instance. VMware's High Availability solution [14] is a typical example of this replication technique. This approach consumes least amount of resources but provides reduced availability than the former methods. Figure 6 illustrates the Markov model of an application for which a dedicated (passive) backup is applied.

A ft_sol can perform replication of a user's application, detection of failures, and recovery from a failed state without requiring any changes to the application's source code. This implies that it is feasible for the service provider to transparently enforce fault tolerance on specified applications. However, based on the failure characteristics of the system, the service provider must derive a deployment level (location of individual replicas) for each ft_sol to correctly realize the fault tolerance service. For the sake of simplicity, we discussed only the availability property of the system in our examples, but similar Markov models can be used to study the reliability property as well.

B. Deployment levels in Cloud infrastructures

Fault tolerance (and resource costs) of an application may vary also based on the location of its replicas. We discuss three different deployment scenarios and identify how fault tree of the service instance can be integrated based on the chosen scenario. A deployment scenario corresponds to the location (or configuration) of the physical host on which individual replicas (VM instances) of an application under a single implementation of a fault tolerance mechanism are created. We assume failures in individual resource type to be independent of each other.

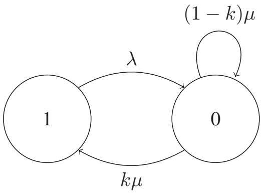

Fig. 6. An example of a Markov model for passive replication

1) Multiple machines within a cluster: Two replicas of an application can be placed on hosts that are connected by a switch (S) i.e., in a LAN. This deployment provides benefits in terms of low latency and high bandwidth but offers least failure independence. Replicas cannot communicate and execute the fault tolerance protocol upon a single switch failure, or a failure in the power distribution unit results in an outage of the entire application. Even worse, if both replicas are placed on the same host, a single component failure will affect both replicas. In this deployment scenario, the cluster level fault tree blocks for each type of resource failure (Figures 1–3) must be connected with a logical AND operator (e.g., DU1, DU2 of the cluster $\wedge$ S connecting the hosts $\wedge$ individual host components). We note that the overall availability and reliability obtained from each fault tolerance mechanism with respect to host failures must be determined using a Markov model.

2) Multiple clusters within a data center: Two replicas of an application can be placed on hosts that belong to different clusters in the same data center i.e., connected via a switch and AggS. This deployment still provides moderate benefits in terms of latency and bandwidth, and offers higher failure independence. The replicas are not bound to an outage with a single power distribution or switch failure. Therefore, to represent the overall availability of an application, in this scenario, the cluster level blocks from the fault trees may be connected with a logical OR operator in conjunction with power and network with an AND operator.

3) Multiple data centers: Two replicas of an application can be placed on hosts that belong to different data centers i.e., connected via a switch, AggS and AccR. This deployment has a drawback with respect to high latency and low bandwidth, but offers a very high level of failure independence. A single power failure has least effect on the availability of the application. In this scenario, the data center level blocks from the fault trees may be connected with a logical OR operator in conjunction with the network in the AND logic.

C. Example of ft_sol behavior at various deployment levels

Since input parameters and availability values of hardware and system software are normally vendor-confidential, we derive this data from the tables published in [8], [10], [11]. Based on this data and using our evaluation scheme, as an

TABLE I
AVAILABILITY VALUES FOR EACH FAULT TOLERANCE MECHANISM IN DIFFERENT DEPLOYMENT SCENARIOS

	Same Cluster	Same Data center, diff. clusters	Diff. Data centers
Semi-Active	0.9871	0.9913	0.9985
Semi-Passive	0.9826	0.9840	0.9912
Passive	0.9542	0.9723	0.9766

example, we derive the overall availability of each representative replication scheme of ft_sols with respect to different deployment levels. Table I illustrates the availability results.

We can see that availability of the application is highest when replicas are placed in two different data centers. The value is slightly lower for the deployment level 2 (replicas in two different clusters) and still lower for scenario where replicas are placed inside the same LAN. Similarly, the overall availability obtained by semi-active replication is slightly higher than semi-passive replication, and lowest for simple passive replication scheme. The values in this table can be used by the service provider to select appropriate ft_sols and its deployment level (see next Section).

V. FAULT TOLERANCE MECHANISMS AND DEPLOYMENT LEVEL SELECTION METHODOLOGY

We introduce a methodology to select appropriate fault tolerance mechanisms and deployment levels based on user's requirements.

A. Matching and Selection Process

We assume that the service provider realizes a range of fault tolerance mechanisms as ft_sols and determines the reliability and availability values that can be obtained using each ft_sol for different configurations (e.g., no. of replicas) and deployment levels (e.g., Table I).

We denote the fault tolerance properties p of a ft_sol using a triple $p{=}(s, \hat{p}, A)$ where s denotes the ft_sol, $\hat{p}$ represents the high level abstract properties such as reliability and availability, and A denotes the set of structural, functional and operational attributes that refers to the granularity at which s can handle failures, benefits and limitations of using s, inherent resource consumption costs and quality of service metrics. Each attribute $a{\in}A$ can take a value, denoted as $v(a)$, from a domain $\mathcal{D}_a$ and a partial ordered relationship $\preceq_a$ can be defined on the domain. For instance, fault tolerance property of a ft_sol s_1 can be denoted as $p{=}(s_1$, {availability=99.995%, reliability=98%}, {mechanism=semi-passive-replication, fault_model=server_crashes, network_faults, power_failures, fault_detection_time=5ms, recovery_time=8ms, n.replica=3}). A hierarchy of fault tolerance properties $\preceq_p$ can also be defined; if P is the set of all properties, and given two properties p_i,$p_j{\in}P$, $p_i{\preceq_p}p_j$ if $p_i{\cdot}\hat{p}{=}p_j{\cdot}\hat{p}$ and $\forall a{\in}A$, $v_i(a){\preceq}v_j(a)$. The attribute values depend on the configuration of the fault tolerance mechanism and values for abstract properties are determined using Markov models and fault trees. A user can specify its requirements in terms of desired abstract properties $\hat{p_c}$ and constraints on attribute values A_c.

Let S be the set of ft_sols available in the system. For a given user request, we first shortlist the ft_sols that satisfy user's abstract property requirements. Let $S'{\subseteq}S$ be the shortlisted set of ft_sols for which $\hat{p_c}{\preceq_p}\hat{p_i}$, $\forall i{\in}S$. Any $s{\in}S'$ can be used to deliver the service if the user does not provide constraints in terms of total resource usage costs or performance of the fault tolerance protocol since high level reliability and availability requirements can be satisfied by any $s{\in}S'$. However, since a user's input may contain specific attribute values, for each ft_sol in S', we compare attribute values for each $a{\in}A$ to obtain a set S'' of candidate ft_sols. In particular, we compare the values of each attribute $v_i(a)$ with the value specified by the user $v_c(a)$, and include those ft_sols in S'' for which $v_c(a){\preceq_a}v_i(a)$. For example, fault_detection_time or recovery_time must be less than or equal to the specified value, whereas number of replicas must be greater than or equal to the specified value. By performing this step, our algorithm selects only those ft_sols that satisfy both user's high level requirements and additional conditions on the attributes. Note that there may be some inconsistencies in the matching and comparison processes that can be handled based on the priorities specified by the user. Finally, we compare each ft_sol within S'' and order them with respect to user's requirements. The first ft_sol in the ordered set S'' is finally used to provide fault tolerance service to user's application since it most appropriately satisfies user's requirements.

B. Illustrative example of matching and selection process

Assume that the service provider realizes three ft_sols with properties

$p_1{=}(s_1$, {availability=99.9%, reliability=99%}, {mechanism=semi-active-replication, n.replicas=3, fault_detection_time=2ms, recovery_time=2ms, deployment_level=2}),
$p_2{=}(s_2$, {availability=95%}, {mechanism=passive-replication, recovery_time=30sec, dimension=shared}), and
$p_3{=}(s_3$, {availability=99.5%, reliability=98%}, {mechanism=semi-active-replication, n.replicas=2, fault_detection_time=4ms, recovery_time=8ms, deployment_level=2})

respectively. If the user requests a fault tolerance support for its banking service, and specifies the following requirements

$p_c{=}(s_c$, {availability≥99%, reliability≥98%}, {fault_detection_time≤5ms, recovery_time≤10ms, n.replicas<3}),

our algorithm first generates the set $S'{=}(s_1, s_3)$ since s_c(reliability)$\leq s_1$(reliability)$\wedge s_c$(availability)$\leq s_1$(availability), s_c(reliability)$\leq s_3$(reliability)$\wedge s_c$(availability)$\leq s_3$(availability). The algorithm then discards s_1 from S' since s_1(n.replicas)$\not<$3; hence, $S''{=}\{s_3\}$. Finally, since $|S''|{=}1$, the ft_sol s_3 is used by the service provider to deliver the fault tolerance service to the user. That is, two replicas of the banking service are created and placed on different clusters within the same data center; the semi-active replication scheme is used to maintain the state of each replica.

Typically, the service provider must realize a two stage delivery scheme: design stage and runtime stage, to deliver high levels of fault tolerance to user's applications. The design stage starts when a user requests a service provider to offer a fault tolerance support to its application. In this stage, the service provider must first analyze the user's requirements, match them with available ft_sols and select an appropriate mechanism to deliver the service. However, since the context of a fault tolerance solution may change at runtime due to the dynamic nature of the Cloud computing environment, the attribute values of each fault tolerance solution offered to the user must be continuously monitored in the runtime stage. For example, the real-time attributes of the host on which a replica is located must be monitored in the runtime stage to ensure that user's reliability requirements are satisfied throughout the life-cycle of the service. The methodology presented in this paper effectively realizes the design stage of the delivery scheme and it can be similarly extended to deal with the runtime stage. We consider developing a more holistic solution with efficient and robust runtime monitoring as part of our future work.

VI. Related Work

Virtualization technology is an important enabler of the Cloud computing paradigm. It allows an infrastructure provider to address concerns related to scalability and availability, and issues with heterogeneous computing resources in data centers. Service availability is often used as the standard metric in service level agreements (SLA). An interesting work presents the availability models for both virtualized and non-virtualized servers in the form hierarchical analytical models [8] and demonstrate encouraging results with the use of virtualization. The system modeling scheme presented in this paper also uses analytical models but it is the only solution that takes the correlation between component failures in typical data centers and deployment scenarios in a large scale system into account. The authors in [11] study the availability attribute in the state-of-art SLA models using similar techniques and conclude that it is beneficial to offer different levels of availability to different user applications. The matching and comparison method presented in this paper, is similar to [2], [7], and allows the service provider to feasibly realize this additional feature. Our proposal enables a service provider to identify suitable fault tolerance techniques when a set of desired properties are given.

An interesting line of research exploits the virtualization technology to improve the availability and reliability of a system. In [3], the authors present a mechanism to continuously "synchronize" the memory state of a node to backup nodes using checkpointing. A fault tolerance middleware that uses the leader/follower replication approach to tolerate crash faults in Cloud computing is presented in [16]. In this paper, we build on the principles of the conceptual framework presented in [6], [7] to utilize the virtualization layer to transparently introduce fault tolerance on applications. We believe that the proposed solution can serve as a basis to holistically realize the perspective of offering fault tolerance as a service in Cloud computing. Security is also an important feature considered in FTM for ensuring dependability (e.g., [1]).

VII. Conclusions

We presented a failure model comprising critical cloud infrastructure resources namely, server components (including VM and VMM), network and power distribution, to analyze the impact of each failure on user's applications. Based on this failure model and representative fault tolerance mechanisms that transparently functions on applications deployed in the VM instances, we discussed suitable deployment contexts and quantified the high level reliability and availability properties for each mechanism. We also presented a methodology to select fault tolerance techniques based on user's requirements. Our future work will mainly focus on extending the models presented in this paper to a larger scale in order to adapt with dynamically changing Cloud computing system attributes.

Acknowledgments

This work was supported by the Italian Ministry of Research within the PRIN 2008 project "PEPPER" (2008SY2PH4), and by the Università degli Studi di Milano within the "UNIMI per il Futuro – 5 per Mille" project "PREVIOUS".

References

[1] C. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati, "A privacy-aware access control system," *Journal of Computer Security (JCS)*, vol. 16, no. 4, pp. 369–392, September 2008.

[2] C. Ardagna, E. Damiani, R. Jhawar, and V. Piuri, "A model-based approach to reliability certification of services," in *Proc. of DEST-CEE'12*, Campione d'Italia, Italy, 2012, pp. 1–8.

[3] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: high availability via asynchronous virtual machine replication," in *Proc. of NSDI'08*, San Francisco, USA, pp. 161–174.

[4] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," *ACM Computer Communication Review*, vol. 41, no. 4, pp. 350–361, 2011.

[5] R. Guerraoui and M. Yabandeh, "Independent faults in the cloud," in *Proc. of LADIS'10*, Zurich, Switzerland, 2010, pp. 12–17.

[6] R. Jhawar, V. Piuri, and M. D. Santambrogio, "A comprehensive conceptual system-level approach to fault tolerance in cloud computing," in *Proc. of SysCon'12*, Vancouver, BC, Canada, 2012, pp. 1–5.

[7] R. Jhawar, V. Piuri, and M. D. Santambrogio, "Fault tolerance management in cloud computing: A system-level perspective," *IEEE Systems Journal*, 2012 (to appear).

[8] S. Kim, F. Machida, and K. Trivedi, "Availability modeling and analysis of virtualized system," in *Proc. of PRDC'09*, Shanghai, China, 2009, pp. 365–371.

[9] G. Koslovski, W. L. Yeow, C. Westphal, T. T. Huu, J. Montagnat, and P. Vicat-Blanc, "Reliability support in virtual infrastructures," in *Proc. of CloudCom'10*, Indianapolis, USA, 2010, pp. 49–58.

[10] W. E. Smith, K. S. Trivedi, L. A. Tomek, and J. Ackaret, "Availability analysis of blade server systems," *IBM Systems Journal*, vol. 47, no. 4, pp. 621–640, 2008.

[11] A. Undheim, A. Chilwan, and P. Heegaard, "Differentiated availability in cloud computing slas," in *Proc. of Grid'11*, Lyon, France, 2011, pp. 129–136.

[12] W. E. Vesely and N. H. Roberts, *Fault Tree Handbook*. Government Printing Office: U.S. Nuclear Regulatory Commission, 1987.

[13] K. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proc. of SoCC'10*, Indianapolis, USA, pp. 193–204.

[14] VMware, "White paper: Vmware high availability concepts, implementation and best practices," 2007.

[15] VMware, "White paper: Protecting mission-critical workloads with vmware fault tolerance," 2009.

[16] W. Zhao, P. Melliar-Smith, and L. Moser, "Fault tolerance middleware for cloud computing," in *Proc. of CLOUD'10*, Miami, USA, pp. 67–74.