High performance fault-tolerance for clouds

Dimosthenis Kyriazis[1], Vasileios Anagnostopoulos[1], Andrea Arcangeli[2], David Gilbert[2], Dimitrios Kalogeras[3], Ronen Kat[4], Cristian Klein[5], Panagiotis Kokkinos[3], Yossi Kuperman[4], Joel Nider[4], Petter Svärd[5], Luis Tomas[5], Emmanuel Varvarigos[3], Theodora Varvarigou[1]

[1]National Technical University of Athens, Iroon Polytechniou 9, Athens, Greece
[2]Red Hat Limited, Cork Airport Business Park, Kinsale Road Cork 700, Cork, Ireland
[3]Computer Technology Institute and Press Diophantus, Patras University Campus, Rio Patras, Greece
[4]IBM Haifa Research Lab, Haifa, Israel
[5]Umeå University, SE-901 87 Umeå, Sweden
dimos@mail.ntua.gr, vanag@mail.ntua.gr, aarcange@redhat.com, dgilbert@redhat.com, d.kalogeras@noc.ntua.gr, ronenkat@il.ibm.com, cristian.klein@cs.umu.se, kokkinop@ceid.upatras.gr, yossiku@il.ibm.com, joeln@il.ibm.com, petters@cs.umu.se, luis@cs.umu.se, manos@ceid.upatras.gr, dora@telecom.ntua.gr

***Abstract*—Cloud computing and virtualized infrastructures are currently the baseline environments for the provision of services in different application domains. While the number of service consumers increasingly grows, service providers aim at exploiting infrastructures that enable non-disruptive service provisioning, thus minimizing or even eliminating downtime. Nonetheless, to achieve the latter current approaches are either application-specific or cost inefficient, requiring the use of dedicated hardware. In this paper we present the reference architecture of a fault-tolerance scheme, which not only enhances cloud environments with the aforementioned capabilities but also achieves high-performance as required by mission critical every day applications. To realize the proposed approach, a new paradigm for memory and I/O externalization and consolidation is introduced, while current implementation references are also provided.**

Keywords—cloud computing; fault-tolerance; high-performance; live-migration; resource consolidation

I. INTRODUCTION

More and more areas of public life become dependent on availability of Internet based services. Banks, logistics, travel, sales and media - to name a few - are severely disrupted when hit by service outages. In August 2012 a lightning strike hit Amazon EC2 Ireland, bringing down Amazon's only European data center for two days [1]. In October 2012 Hurricane Sandy hit New York City requiring the partial shutdown of New York City power grid forcing several high-traffic websites off the Internet, with far reaching effects on North-American as well as European users [2]. Outages have serious implications on the continued operation of businesses - causing direct loss of revenue, legal liabilities as well as long term damage to reputation and brand name: a recent survey estimated a staggering 20B Euro annual revenue loss caused by IT downtime [3]. Similar outages have occurred in 2013, affecting the service provision of Google Drive, Dropbox, VISA, and Windows Live amongst others [4].

However, mitigating the effects of downtime requires significant investment with meticulous planning to appropriately address each type of common downtime cause. While unrecoverable software faults can either be addressed via application-specific improvements or via generic watchdog solutions, other causes such as unplanned hardware faults or planned maintenance are mainly addressed by expensive hardware-level approaches. Current solutions are either application specific, limited to uni-processor workloads, and lacking required performance targets, or require proprietary hardware.

Aiming to address the needs of mission-critical services, in this paper we introduce an architecture enabling advanced high performance fault-tolerance and disaster recovery in cloud environments. Core in the proposed architecture is a new paradigm of virtualized resource consolidation in which memory and I/O resources used by a guest Virtual Machine (VM) are provided by multiple external hosts. What is more, the proposed reference architecture is not limited to single cloud data centres but addresses cross-site scenarios, given that advances in network fabrics technology have brought to significant reduction in network latencies [5], [6], [7]. The proposed paradigm of resource consolidation provides the enabling ground to facilitate high performance fault tolerance given that existing virtualization-based fault tolerance approaches, such as Kemari [8] and Remus [9], incur major performance penalties due to VM check-pointing and state synchronization of the passive VM in an active-passive topology; while the alternative approach of lockstep processing in a primary-secondary topology offered by VMWare FT [10] has no support for Symmetric Multi-Processing (SMP) guest VMs. One way of reducing the cost of Kemari's state synchronization, without impairing the generality of the solution, is to reduce, by means of virtualized resource consolidation, part of the VM state. A different implementation but with the same goal of reducing VM state was demonstrated by Ohmura in [11] showing that Kemari's performance for fault tolerance when combined with I/O logging (on the active VM) and replay (on the passive VM) achieved a 50% reduction of network bandwidth on a file I/O benchmark (e.g. IOZone).

Externalization (from the perspective of the guest VM) and consolidation of virtualized I/O will be carried out by transferring I/O operations requested by the VMs to a

978-1-4673-7194-0/15/$31.00 ©2015 IEEE

dedicated remote server responsible for executing the requests. For consolidation of virtualized I/O we propose combining SR-IOV [12], [13] which provides near native I/O performance, together with paravirtual I/O technologies [14], [15] which enable I/O interposition (required for fault tolerance). In terms of implementation, externalizing and consolidating memory of guest VMs we focus on extending the Linux kernel with user-space page fault support, which can be exploited to implement a networked accessible memory pool.

The remainder of this paper is structured as follows: Section 2 shortly presents related work with respect to fault tolerance including I/O and memory virtualization as the main enablers. Section 3 introduces the main capabilities of the proposed reference architecture, which is detailed in Section 4. Initial results are provided thereafter, while Section 5 concludes the paper with a discussion on future steps.

II. Related Work

With respect to I/O virtualization, while several performance issues have been addressed, the biggest remaining virtualization-overhead problem is for I/O-bound workloads, i.e. workloads which are network- or disk- intensive. Three major techniques are common for hosts to virtualize I/O services for their guests, emulation [14], where a familiar device (e.g. a common network card) is emulated, paravirtualization [14], [15], which emulates a new device, designed not to resemble any existing device but to be as efficient as possible when used across the guest-host boundary, and device assignment [16], [17], where the host gives a guest (mostly) direct access to a portion of a certain physical device. In terms of performance, the device assignment approach depicts the best one [17], and it has become even clearer when Exit-Less Interrupt (ELI) [18], designed for device assignment, achieved the coveted bare-metal performance for I/O workloads. ELI removed most of the virtualization overhead by eliminating hypervisor involvement (a.k.a. exits) in handling of interrupts from the assigned device. With such ELIs, the entire I/O critical path became exit less, and could therefore proceed at bare-metal speeds. Despite the proven performance advantage of device assignment, paravirtualization is preferred or even required because device assignment does not support I/O interposition and thus cannot be used when the hypervisor needs to intercept all the I/O channels to implement fault tolerance capabilities. Device assignment also requires more expensive hardware (an IOMMU [19] and, if the same device is to be assigned to several guests, SRIOV-supporting devices [12], [13]). For these and other reasons, most real-world applications of virtualization today choose to use paravirtual I/O. Using paravirtual I/O, the hypervisor running on each server is responsible for interposing on the I/O of each of its guests. The hypervisor requires and consumes physical resources such as CPU, RAM, and SSDs from each of the servers which could otherwise be assigned to the running guests or be used to run additional guests. In addition, this model degrades the performance of I/O intensive guests and limits the scalability of the system [20].

With respect to fault tolerance, VMWare High Availability [21] addresses two kinds of failures, an OS/Application failure by restarting the VM on the same physical server and a physical server failure by restarting the VM on a different physical server which has access to shared VMFS clustered file system image of the VM. Amazon AWS EC2 provides a comparable level of reliability by recovering from a physical server failure by restarting the EC2 compute instance from the AMI image, granting at least 99.95% up time during a service year [22]. Notable that the application has to be adapted to AWS platform, especially in respect to storing all state on either AWS S3 storage or on other comparable means. NetApp MetroCluster [23] is a software solution which provides disaster recovery combining array-based clustering with synchronous mirroring delivering continuous availability and zero data loss. Via integration with hardware and virtualization platforms it enables a transparent fail-over in case of storage outage, zero application downtime, or application interruption when combined server redundancy solutions such as VMWare FT. Additional application specific approach refer to ClusterXL security appliance by CheckPoint [24], which is based on fully redundant hardware in an N+N schema, complemented with internal state shipping from the active server to the passive server. As internal state tables may be large in volume this solution can only be deployed in a LAN and requires a dedicated NIC for handling state transfer. As this solution is implemented in the application schema it is hardware agnostic (supporting any physical of virtual hardware with full Symmetric Multi Processing support).

Server lockstep fault tolerance is another approach employed by many enterprises that require a high degree of availability. Reference implementations include ftServer from Stratus [25] and NEC Express5800 [26], which use a technique called Lockstep Processing in which the server is outfitted with Dual Modular Redundancy (DMR) and allows the redundant components to process the same instructions simultaneously. Comparable to ftServer is VMWare's FT/vLockstep [27] hypervisor based software-only solution, which relies on deterministic record/replay of non-deterministic inputs. The primary and the secondary VM share the same virtual disk on shared storage, but all I/O operations are performed only on the primary host. While the hypervisor does not issue I/O produced by the secondary, it posts all I/O completion events to the secondary VM at the same execution point as they occurred on the primary. Stratus everRun-MX [28] provides a similar software-based fault tolerance solution specifically addressing the windows OS.

Finally, server state synchronization approaches for fault tolerance, such as Remus [9] and Kemari [29], [8] provide high-availability, using efficient N+1 topology in which one physical server can act as a warm backup (a host with stand-by VMs to serve as needed) of N active physical servers. These novel techniques are different from the previously mentioned Lockstep Processing as the active VM is the only VM to execute operations. The passive or backup VMs (which reside in a backup server) get periodic state synchronization messages, which ensure their state is consistent with any I/O output events which were sent out by the active VM -ensuring that in the event of a hard fault on the active host (running the active VM), a seamless fail-over can be done to the backup host (running the passive VM) with no client-noticeable state change.

Comparing to the approaches presented in this section, the proposed reference architecture focuses on high performance and decreased network overheads (no need for heavy weight state tracking synchronization), while being application agnostic.

III. Core Capabilities

In this section we introduce the core capabilities / features of the proposed reference architecture. The capabilities include outcomes with respect to the enabling technologies (e.g. I/O consolidation), techniques targeting fault tolerance (e.g. modelling and semantics), as well as the required enrichments to cloud middleware to incorporate the proposed functionality (e.g. OpenStack enhancements).

A. I/O consolidation

I/O consolidation is a new paravirtual I/O model providing a centralized, scalable facility for handling I/O services. It does so by decoupling I/O from computation on the machines hosting the VMs, and shifts the processing of the I/O to a dedicated server (I/O hypervisor). Firewall, DPI (deep packet inspection) and block-level encryption are examples of such I/O services. These I/O services can consume a lot of CPU resources, thus, consolidating them in a dedicated server increases CPU utilization and accommodates changing load conditions where demand from different hosts fluctuates.

B. I/O scheduling

An important operation of the I/O consolidation is the way the I/O operations, originating from multiple VMs in several physical hosts, are scheduled for execution in the hypervisor's available cores. To this end we will develop and evaluate a number of online scheduling algorithms for the I/O hypervisor. The main objectives of such a scheduling algorithm are the minimization of I/O blocking and the maximization of the I/O throughput and resource utilization. I/O scheduling also highly affects the efficiency of the I/O hypervisor in terms of the number of VMs that can be served concurrently.

C. Memory externalization

There are some use cases where the contents of parts of guest physical memory are not yet present on the host running that VM. Allowing the hypervisor to efficiently manage faults from the VM as it tries to access this memory, and then fetch the data from other hosts is 'memory externalization'. Existing techniques of using page-protection to mark areas of memory as faulting, and mmap to remap those areas are expensive. Existing Linux kernel modifications have been very specific and interfere with other kernel memory optimisations such as huge pages and page sharing. The current work provides an efficient method for userspace applications (in this case the QEMU hypervisor component) to detect accesses to missing pages (i.e. userfault), and to place the data for those pages as it arrives.

D. Post-copy VM migration

Existing (pre-copy) migration schemes may fail where the VM being migrated changes memory more quickly than that memory can be examined and transmitted to the destination host. This is a major problem when migration is used for host evacuation in terms of an emergency (e.g. running on time limited backup power in a data-centre) since the transmission time is unbounded. Post-copy migration solves the problem by allowing the CPUs on the destination to start executing before all the memory has been transferred. This provides a demonstration of the memory externalization capabilities that are used to satisfy the destination CPUs memory accesses on a demand basis, limiting page transmission to a single copy of the page and thus bounding the total migration time.

E. VM fault tolerance

The current capability refers to the implementation of a full fault tolerance approach between a pair of hosts. While, the particular technique to be used in order to realize this feature is still under investigation (one of the alternatives being examined is state check-pointing), however it will allow survival of unmodified applications in the event of host failure.

F. Fault modelling and semantics

Fault modelling has been limited in many cases in the form of fail-stop faults (the simplest) and Byzantine faults (the most challenging). Link faults are re-factored as process faults in many studies. The duration of faults is also something to be taken into consideration because it allows fast recovery. Proactive fault modelling and prediction [30] poses many more modelling challenges. In this case the server must advertise various parameters that can be correlated with the possibility of a fault. In this case many optimizations to a distributed fault detection or consensus protocol can be applied in order to improve the reliability of the system in question. The proposed feature extends the aforementioned techniques to enable proactive fault modelling by selecting the available metrics that are suitable for a reliable fault prediction. These models will be incorporated into a distributed fault detection service providing improvements over APIs like the ones presented in [31].

G. Fault detection

Fault detection in a Metropolitan Area Network (MAN) is a problem that has been studied in the literature during the past thirty years, see for example [32] and references therein. However, high-performance fault-tolerance poses many challenges because of the deployment in a metropolitan area network (e.g. use of multi-site cloud environments) and the maintenance of a distributed RAM pool. While keeping a heartbeat service is a typical way of detecting faults the dynamic nature of Internet pauses various problems. The biggest is the unpredictable nature of the delays. Having a good estimate of a lower bound is crucial because in the case of leader election protocols like Raft the number of leader election periods can become prohibitively large. The consequence is that the protocol is rendered useless. To realize this feature, one approach that is currently implemented is based on a Bayesian framework. More specifically the intractability of exact inference to large, complex networks drives us to the investigation of approximate inference algorithms, which is a compromise between speed and accuracy. A series of existing approaches like [33] will be

extended to enable proactive fault detection and efficient communication of faults and corrective actions within and across data-centres.

H. Traffic redirection

Fast and transparent redirection of traffic between the active and the backup VMs is necessary so as to achieve fault tolerance in clouds. In a LAN environment, typically in a data-centre, were both VMs are in the same domain this can be relatively easily achieved. On the other hand, traffic redirection in a MAN/WAN environment is more challenging than in LAN, since active and backup VMs are hosted in different data-centres and IP domains. This feature aims at providing techniques in various network layers enabling traffic redirection in a MAN/WAN environment. In particular, BGP-based methods (e.g. using anycast), which take control of the network through Software Defined Networking approaches [34] as well as the HAProxy solution that offers high availability, load balancing for TCP and HTTP-based applications [35] are exploited to realize the traffic redirection feature.

I. OpenStack enhancements

OpenStack [36] has been selected as the platform of choice due to its wide deployment and is supported both by major industry players as well as an active open source community. The main goal is to integrate the proposed features like resource consolidation and fault tolerance into a cloud management platform, and thus enable their exploitation. As described earlier, fault tolerance is commonly provided via replication of VMs, where a redundant secondary VM is used to recover from failures of a primary VM, also called active-passive replication [37]. Through memory and I/O consolidation the efficiency of VM pairs synchronization and role switching is improved. They, however, have to be enabled within the OpenStack framework. To this end, the following capabilities are provided by the proposed architecture:

- *Multi-VM scheduling*: A new scheduling algorithm considers both active and passive VMs during the scheduling phase, and not only avoids co-locating VM pairs in the same host, but also considers other constraints, such as the trade-offs between VM distribution (fault-tolerance resilience) and resource fragmentation (synchronization overhead). In this context, network-awareness [38] becomes quite important. Minimizing the communication traffic between VMs by placing them wisely inside the network is important, but also considering possible network or hosts failures. Therefore, in order to achieve high availability and increase fault tolerance in cloud services with redundant VMs, the scheduler also considers placing these redundant VMs in different fault domains on disjoint hardware in the network, such as different switches.
- *Volume replication*: OpenStack Cinder will be extended since it is the component that manages block storage. Enabling Cinder to manage replication of volumes is a key capability toward recovery of an application workload in an alternative data centre in a MAN or WAN distance.
- *Workload disaster recovery*: Recovering a workload in an alternative data centre raises the need to recreate the same running environment as in the primary data centre. This includes the data (see above), the application, and how it is being run in the cloud. This feature enables the administrator of a cloud environment to seamlessly recover the application in a automatic manner.

IV. Architecture

Based on the aforementioned capabilities, this section introduces the proposed architecture. An overview is initially provided, while a more detailed discussion on the main architectural components follows.

A. Overview

The following figure presents the main building blocks of the proposed architecture. The left column focuses on the building blocks for data centre high availability and the right column focuses on metro and wide area disaster recovery.

The building blocks are partitioned into layers: the I/O layer, including I/O consolidation; the memory layer, including externalization of memory and VM migration; cloud management, including multi VM placement and support for volume replication in OpenStack; the fault tolerance layer, including modelling, semantics, detection and both VM and workload fault recovery; the networking layer, including network redirection; and the dashboards in the user experience layer.

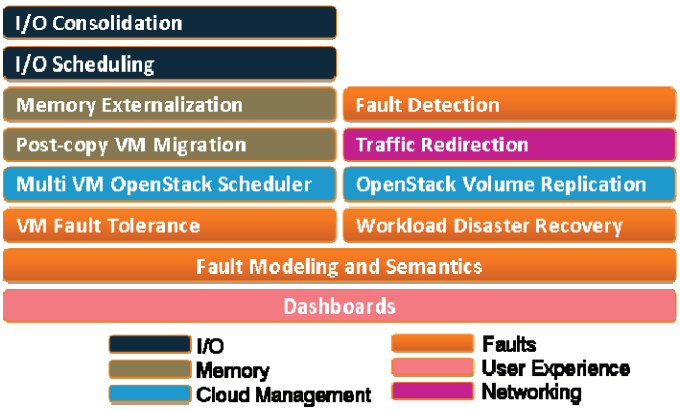

Fig. 1. Architecture overview

B. Main components

The main components realizing the fault-tolerance architecture include the following:

- **Split I/O Ethernet Transport**: Existing transport protocols such as TCP/IP incur high overhead. To achieve maximum performance we implemented our own protocol over layer 2 (Ethernet) to enable efficient and scalable communication between the Split I/O front-end and Split I/O back-end components. With

the gain in performance, we lose the features provided by such protocols. However, we don't need most of the features (such as flow control) since we are on a dedicated Ethernet segment. We depend on the guest's networking stack to handle error detection and correction. Typically the MTU (Maximum Transition Unit) size of Ethernet is 1500 bytes. Since we have exclusive use of the Ethernet segment we are not constrained by the typical MTU size. By using a larger MTU we reduce the number of fragments created, thereby having fewer packets to process at the receiving end.

- **Split I/O Generic Front-End**: Provides common Split I/O data services to both the network and block front-ends. This is merely a modular design decision, taking the common parts and putting them in a shared module.
- **Split I/O Generic Back-End**: Provides common Split I/O data services to both the network and block back-ends. It's also responsible for exposing a control interface to the cloud management layer that can be used to link the VMs with the corresponding I/O hypervisor and manage the virtual network and virtual block devices. The back-end is responsible for multiple virtual devices running on multiple VMs. These virtual devices generate many I/O requests that are offloaded to the back-end for processing. To reduce the latency and to maximize CPU utilization we turn off interrupts coming from the Ethernet Network Interface Controller - NIC (that serves as the channel between the two ends) and instead, we poll on the device for incoming requests. A dispatcher load balances the work among all the available CPUs.
- **Split I/O Block Front-End**: This component is responsible for exposing virtual block devices (disks) to the guest operating system. All read/write requests are transmitted to the block back-end running in the I/O hypervisor for further processing. There is no architectural limit to how many virtual block devices a VM can have. A virtual block device is created and exposed to the guest OS only when told by the back-end. Each block request that the OS generates is treated similar to the way virtio handles these requests, but instead of placing them on a shared ring buffer the request is transmitted over the network to the back-end for further processing. Since the guest's block I/O stack relies on guaranteed delivery, the front-end driver implements a retry mechanism in case of packet loss of the underlying fabric.
- **Split I/O Net Front-End**: This component is responsible for exposing virtual network devices to the guest operating system. The virtual NIC is created to the request of the I/O Hypervisor. This virtual NIC is a “first class citizen”, it has the same capabilities and features of a physical Ethernet NIC. Each L2 frame generated by the guest OS is transmitted to network back-end running in the I/O hypervisor for further processing.
- **Split I/O Block Back-End**: This component is responsible for processing the block read and write requests sent by the front-end running within each VM. It maps the (remote) virtual block device exposed to each VM with a (local) block device accessible by the I/O hypervisor. The backing block device can be, for example: local SSD or a remote SAN disk. Various services can be applied to the I/O request before committing it to the backing device, for example: block level encryption, anti-virus scanning, deduplication, etc.
- **Split I/O Net Back-End**: This component is responsible for receiving/sending virtual network L2 frames from/to the virtual network devices of each VM. It bridges the (remote) virtual network devices exposed to each VM with a (local) tap/macvtap interface. The tap interface can be connected with any virtual network (e.g. OVS/Linux Bridge) and the macvtap interface can be connected to any physical NIC. Various services can be applied on the I/O request before sending it to its destination, for example: firewall, Deep Packet Inspection (DPI), etc.
- **Split I/O Fault tolerance**: This component implements the interceptor software design pattern to intercept all the I/O traffic coming from the Split I/O generic back-end. The component buffers all the I/O requests before they are submitted to the concrete network and block layers for processing and release the buffers once the VM fault tolerance check-pointing mechanism confirms the active and passive VMs were successfully synchronized.
- **In-Memory Block I/O Consolidated Cache**: Provides consolidated remote memory caching services for the virtual block devices exposed to the VMs. The block data is cached in the I/O hypervisor RAM. Using this consolidated remote block cache mechanism we can reduce the amount of local RAM commonly allocated by each VMs to cache block data.
- **Kernel extensions for memory externalization:** These components provide efficient mechanisms for the userpsace hypervisor to detect missing pages and map them in.
- **QEMU extensions for post-copy live migration:** A post-copy implementation (based on the above kernel extensions) providing a transport for page requests and the sequencing to perform the migration.
- **Fault tolerance mechanisms:** They provide synchronisation of the state between a VM on two hosts, and the interfaces to the other components to trigger failover and ensure synchrony of the IO.
- **Nova-scheduler modification**: The new scheduler provisions two VMs instead of one. It sends the allocation request to two instances of nova-compute running on different hosts, indicating which one is the active VM and which one is the passive one. After the VMs has been provisioned, it communicates with

nova-IORCL module (described in the following paragraphs) to attach network and storage devices (through OpenStack Neutron and Cinder, respectively) to the I/O hypervisor, and to establish the connection between the VMs and the I/O devices through the I/O hypervisor. The new scheduler also incorporates novel scheduling algorithms for better placement decisions regarding VM pairs, allowing a trade off between fragmentation and distribution, and to optimize the allocation over time.

- **Nova-compute modification**: This component is modified, firstly, to move the Cinder and Neutron calling process to nova-IORCL, so that I/O devices (network and storage) are not attached to the VMs. Secondly, it communicates with the modified host hypervisor to indicate whether the VM is active or passive, and to provide the needed flags for memory synchronization between both VMs.
- **Nova-IORCL module**: This is a new nova module in charge of managing the I/O hypervisor through the interface provided to control the split I/O Generic backend. It handles the Cinder and Neutron calls originally located at nova-compute with the objective of allocating the network and storage devices to the I/O hypervisor instead of to the hosted VMs. Once this is done, it communicates with the I/O hypervisor to connect the VMs to their associated I/O devices.
- **Cinder and Neutron**: These modules must communicate with nova-IORCL to attach/detach virtual devices to the I/O hypervisor. Additionally, other small modifications may be needed for storing fault tolerance information (nova-database) and for communications purposes between working and controlling OpenStack services, such as nova-compute and nova-conductor.
- **Libvirt**: OpenStack uses Libvirt as a middle layer to communicate with the hypervisor, which in the proposed architecture is a modified version of Qemu-KVM. To support the proposed features, Libvirt is extended to allow support for post-copy live migration.
- **Fault-detection mechanism:** This component enables failure detection in metro-area cases (i.e. between data-centres) and realizes a proactive fault detection distributed algorithm (based on RAFT algorithm) in order to detect and communicate efficiently faults. Specific fault semantics are prioritized and weighted in order to identify faults in a proactive way based on the corresponding real-time monitoring information.

C. *Components interaction*

Figure 2 below describes how the architecture deals with a fault that can be recovered with the data center. Upon detection of a fault, the VM fault tolerance component of the cloud management promotes the passive VM to be active and updating the device mapping that resides in the I/O hypervisor. This provides high availability and seamless business continuity of the running application.

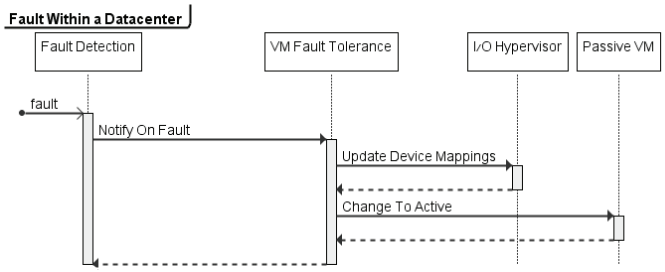

Fig. 2. Components interaction for faults within a data center

In case of a data centre scale disaster, the architecture utilizes the disaster recovery mechanism, which on fault makes the application data volume available in the secondary cloud, deploy the VM which is running the application, and complete the recovery by interacting with the traffic redirection mechanism.

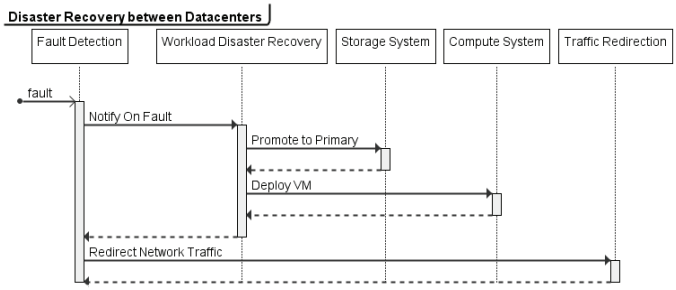

Fig. 3. Components interaction for disaster recovery in a metro-area network

D. *Initial results*

The presented reference architecture is currently being developed within the framework of the EU-funded ORBIT Project [39]. Initial implementations of the post-copy live migration capabilities and the kernel userfault are available online [40], [41]. The initial results capture the latency in providing a page still resident on the other system in a migration case. This test was performed using a 10Gb ROCE link (using TCP rather than RDMA) on a pair of systems with 2xXeon E5-2407 2.2GHz CPUs (8 cores total per system) and 24GB RAM. The VM being migrated consisted of an 8GB Linux image with 6 cores running Google Stress Apptest continuously modifying memory, and thus wouldn't be able to migrate using standard precopy migration. The basic RTT is ~300us. The common latencies are around 10ms.

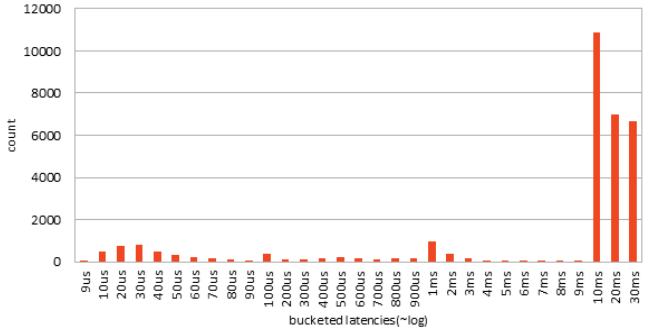

Fig. 4. Latency measurements

It should be highlighted that the proposed features and implementations, such as the post-copy live migration can be exploited independently of the main goal of the proposed approach (i.e. fault tolerance). For example, the post-copy live migration and the kernel userfault implementations provide enhancements for volatile pages in other cases as well such as Android operating systems or Firefox browser.

V. CONCLUSIONS

The proposed reference architecture aims at addressing the cross-industry need for more cost-effective ways to ensure business continuity by minimizing downtime and outages. It realizes an innovative paradigm for memory and I/O externalization and consolidation in order to enable high performance fault-tolerance in cloud environments. Notwithstanding, it is within our future plans to further develop and evaluate the proposed architecture through real-world scenarios including different classes of applications (e.g. data-intensive, mission critical, computation-oriented, etc).

ACKNOWLEDGMENT

The research leading to the results presented in this paper has received funding from the European Union's 7th framework programme (FP7 2007-2013) Project ORBIT under grant agreement number 609828.

REFERENCES

[1] Lighting strike zaps EC2 Ireland. http://www.zdnet.com/blog/saas/lightning-strike-zaps-ec2-ireland/1382.

[2] Hurricane Sandy knocks out NYC data centers: Websites, services down. 2012. URL: http://www.zdnet.com/hurricane-sandy- knocks-out-nyc-data-centers- websites-services-down-7000006588/

[3] IT Downtime Costs $26.5 Billion In Lost Revenue. http://www.informationweek.com/storage/disaster-recovery/it-downtime-costs-265-billion-in-lost-re/229625441

[4] Worst Cloud Outages of 2013, http://www.infoworld.com/slideshow/107783/the-worst-cloud-outages-of-2013-so-far-221831#slide1

[5] Latency on a Switched Ethernet Network. http://www.ruggedcom.com/pdfs/application_notes/ latency_on_a_switched_ethernet_network.pdf. 2008.

[6] Is 10 Gigabit Ethernet Ready for HPC. http://www.hpcwire.com/hpcwire/2010-11-18/is_10_gigabit_ethernet_ready_for_hpc.html. 2010.

[7] Philip J. Sokolowski. “Performance considerations for network switch fabrics on linux clusters”. In: In Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems. 2004.

[8] Yoshi Tamura. “Kemari: Fault Tolerance VM Synchronizaiton based on KVM”. 2010. http://www.linuxkvm.org/wiki/images/0/0d/0.5.kemari-kvm-forum-2010.pdf.

[9] Brendan Cully et al. “Remus: high availability via asynchronous virtual machine replication”. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation. NSDI’08. San Francisco, California: USENIX Association, 2008, pp. 161–174.

[10] Ganesh Venkitachalam Daniel J. Scales Mike Nelson. The Design and Evaluation of a Practical System for Fault-Tolerant Virtual Machines. Tech. rep. VMWare Inc.

[11] Kei Ohmura. “Rapid VM Synchronization with I/O Emulation Logging-Replay”. http://www.linuxkvm.org/wiki/images/5/5c/2011-forum-logging-replay.pdf.

[12] PCI SIG. Single Root I/O Virtualization and Sharing 1.0 Specification. 2007.

[13] Yaozu Dong, Zhao Yu, and Greg Rose. “SR-IOV networking in Xen: architecture, design and implementation”. In: 2008.

[14] J. Sugerman, G. Venkitachalam, and B.H. Lim. “Virtualizing I/O devices on VMware workstation’s hosted virtual machine monitor”. In: Proceedings of the General Track: USENIX Technical Conference, 2002

[15] Rusty Russell. “virtio: towards a de-facto standard for virtual I/O devices”. In: 42.5 (2008), pp. 95–103.

[16] K. Fraser et al. “Safe hardware access with the Xen virtual machine monitor”. In: 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS). 2004.

[17] Ben-Ami Yassour, Muli Ben-Yehuda, and Orit Wasserman. Direct Device Assignment for Untrusted Fully-Virtualized Virtual Machines. Tech. rep. H-0263. IBM Research, 2008.

[18] Abel Gordon et al. “ELI: Bare-Metal Performance for I/O Virtualization”. In: 2012.

[19] Darren Abramson et al. “Intel Virtualization Technology for Directed I/O”. In: 10.3 (2006), pp. 179–192.

[20] Abel Gordon et al. “Towards Exitless and Efficient Paravirtual I/O”. In: International Systems and Storage Conference (SYS-TOR). 2012.

[21] VMWare. VMWare High Availability (HA). Tech. rep. VMWare Inc. URL: http://www.vmware.com/pdf/ha _ datasheet.pdf.

[22] AWS. Amazon EC2 Service Level Agreement. http://aws.amazon.com/ec2-sla/.

[23] NetApp MetroCluster. http://www.netapp.com/us/products/protection-software/metrocluster. aspx.

[24] ClusterXL: Software-based High Avialability and Load Sharing. http://www.checkpoint.com/products/ clusterxl/index.html.

[25] Stratus. Stratus ftServer Architecture. Tech. rep. Stratus Inc.

[26] NEC. Fault Tolerant Server. http : / / www . necam . com / servers / enterprise / doc . cfm ? t = 5800A1080a

[27] VMware vShere 4 Fault Tolerance: Architecture and Performance. http://www.vmware.com/files/pdf/ perf-vsphere-fault_tolerance.pdf.

[28] everRun MX Software: Always-on Application Availability For Windows Environments. Tech. rep. Stratus Inc.

[29] Yoshi Tamura. “Kemari: Virtual Machine Synchronization for Fault Tolerance using DomT”. In: 2008.

[30] Felix Salfner, Maren Lenk, and Miroslaw Malek. 2010. A survey of online failure prediction methods. ACM Comput. Surv. 42, 3, Article 10

[31] Paul Stelling, Cheryl DeMatteis, Ian Foster, Carl Kesselman, Craig Lee, Gregor von Laszewski ,"A fault detection service for wide area distributed computations", Cluster Computing 09-1999

[32] Jian Dong, Decheng Zuo, Hongwei Liu, Xiaozong Yang ,"DPHM: A fault Detection Protocol based on Heartbeat of multiple Master-nodes", Journal of Electronics (China), 24(4):544-549, 2007

[33] Dong, HJ;Qiu, XS;Cheng, L;Li, ZQ;Qiao, Y,"Fast Fault Localization for Large-scale IP-based Communication Systems Using Bayesian Networks", Chinese Journal of Electronics 18(4), 735-740, 2009

[34] A. Gämperli, V. Kotronis and X. Dimitropoulos, "Evaluating the Effect of Centralization on Routing Convergence on a Hybrid BGP-SDN Emulation Framework", ACM SIGCOMM, 2014.

[35] HAProxy, http://www.haproxy.org

[36] OpenStack, https://www.openstack.org

[37] G. Venkitachalam et al. “The design and evaluation of a practical system for fault-tolerance virtual machines”, In: Operating Systems Review, vol.44, no 4, pp 30-39, 2010.

[38] O. Biran et al. “A stable network-aware VM placement for cloud systems”, In: CCGrid, pp 498-06, 2012

[39] ORBIT Project, http://www.orbitproject.eu/

[40] Post-copy implementation, http://lists.gnu.org/archive/html/qemu-devel/2014-07/msg00842.html

[41] Kernel userfault implementation, http://lists.gnu.org/archive/html/qemu-devel/2014-07/msg00525.html