# Toward a High Availability Cloud: Techniques and Challenges

Cuong Pham, Phuong Cao, Zbigniew Kalbarczyk, Ravishankar K. Iyer
Center for Reliable and High-Performance Computing
University of Illinois at Urbana-Champaign
1308 W. Main Street, Urbana, IL 61801, USA
{pham9, pcao3, kalbarcz, rkiyer}@illinois.edu

*Abstract* — **Cloud computing in its many forms has become the key computing-infrastructure that supports business and more recently governmental computing across the globe. With its geographical spread and value proposition comes the need to provide guaranteed level of availability in the infrastructure and in its services. Multicore processing, virtualization, distributed storage systems and an overarching management framework that enable a *Cloud*, offer a plethora of possibilities to provide high availability using commodity systems. Herein lie the opportunities and challenges discussed in this paper.**

***Keywords-component: Cloud Computing, Multicore, Virtual Machine, Availability***

## I. INTRODUCTION

Cloud computing has become increasingly important to many businesses and organizations. However, delivering a higher level of availability has been one of the biggest challenges for this computing paradigm. As this new technology is deployed on the large scale, the likelihood of incidents has progressively increased with the last few years witnessing several spectacular incidents (as exemplified in section II).

Multiple technologies such as *multicore processor architectures*, *virtual machines*, and *storage systems* have emerged as enabling technologies for cloud computing. Cloud computing is a model that truly brings-in the advances in these technologies to provide computing as a service. Multicore processors and storage system are used to build the cloud computing infrastructure, whereas the virtual machines form the basis for the cloud computing platform.

This paper briefly reviews and critiques the state-of-the-art techniques for enhancing the dependability of cloud computing and provides our view of the challenges in deploying high-availability cloud infrastructures. We associate the challenges with different layers in a multi-layered cloud architecture. The key challenges outlined in the paper include the following:

(i) ***Multicore processors*** potentially bring in natural redundancy at the processing level. Several standard techniques, e.g., turning off a faulty core, static and dynamic sparing have been proposed to exploit to tolerate hard and soft errors. However, many of such techniques, e.g., core salvaging, are papers or simulated designs and need to be thoroughly validated in the context of the value they bring to the overall cloud infrastructure.

(ii) ***Virtualization*** introduces an additional separation between the low-level hardware and high-level applications. However, it also brings new failure modes to the interfaces between the hypervisor, hardware, and operating system, e.g., error propagation between guest system and the hypervisor. Directed effort is required to ensure fault/error containment and checkpoint consistency if checkpointing is used to keep the VM snapshot for rapid recovery.

(iii) ***Data storage*** driven availability continues to be a significant challenge. With the wide geographical spread of the *cloud*, opportunities for partitioning and inconsistency in the data are not in-significant. While approximation methods are used to achieve data consistency and reducing the performance impact, there is need for sound approaches to determine the tradeoff between consistency and availability in real environments.

(iv) ***Cloud infrastructure and service assessment*** and validation methods are greatly lacking. The cloud's massive scale demands metrics and sound methods for measurement, modeling, and analysis to obtain trusted assessment results.

## II. CLOUD OUTAGES

We start our discussion by highlighting representative examples of cloud outages. These outages did not only affect the cloud availability, they also forced cloud providers to make important design changes in the cloud infrastructure. These are the valuable lessons for constructing the next generation of highly available and secure clouds.

### *A. Microsoft Azure*

During a routine operating system upgrade on March 13, 2009, the deployment service within Windows Azure began to slow down due to networking issues. This caused a large number of servers to time out and fail.

Applications running only as a single instance (i.e., without replication) shut down when the corresponding server went down. Very few applications running multiple instances failed, although some were degraded due to one instance being down. In addition, the ability to perform management tasks from the web portal appeared unavailable for many applications due to the Fabric Controller being loaded with work during the serialized recovery process.

To prevent such occurrences in the future, Microsoft has been fixing the network issues. It is refining and tuning recovery algorithms to ensure handling malfunctions quickly and gracefully. For continued availability during upgrades, application owners are encouraged to deploy their application with multiple instances. The second instance of an application is not counted against quota limits to allow customers to run two instances of each application.

**978-1-4673-2266-9/12/$31.00 ©2012 IEEE**

TABLE 1: SUMMARY OF TECHNIQUES AND CHALLENGES THAT AFFECT CLOUD DEPENDABILITY

| Cloud layer | Dependability Techniques | | Comments |
|---|---|---|---|
| **Multicore Processor** | Permanent failure tolerance<br>• Core disabling<br>• Core salvaging: at ISA, microarchitecture, or software layer | Transient error tolerance: DMR or TMR using<br>• Static core binding<br>• Dynamic core binding | • Need techniques to cope with both hard and transient errors<br>• Introduces complexity in the architecture<br>• Limited use in cloud (using commodity hardware)<br>• High error rate due to high CPU utilization |
| **Virtual Machine** | • Monitoring at hypervisor layer<br>• VM Checkpoint and Rollback<br>• VM Migration | | • New failure modes at hypervisor<br>• Failure propagation: guest to host; host to management system<br>• Minimize checkpoint corruption and overhead<br>• Semantic gap between hypervisor and guest system |
| **Storage System** | • Amazon Dynamo<br>• Google Datastore (uses Paxos to support replication) | | • Trade-offs between consistency and availability<br>• Testing/benchmarking challenges |

### B. *Amazon S3*

The 8-hour outage of Amazon services on July 20, 2008, was caused by a single bit error in messages communicated (using a gossip protocol) between the servers. In their postmortem analysis, system engineers at Amazon determined that "*there were a handful of messages ... that had a single bit corrupted such that the message was still intelligible, but the system state information was incorrect. We use MD5 checksums throughout the system, for example, to prevent, detect, and recover from corruption that can occur during receipt, storage, and retrieval of customers' objects. However, we didn't have the same protection in place to detect whether this particular internal state information had been corrupted. As a result, when the corruption occurred, we didn't detect it and it spread throughout the system....*" [1].

Amazon decided to add one more layer of checksum to protect the stored internal state information.

### C. *BitBucket*

On October 3, 2009, BitBucket (https://bitbucket.org/) experienced 16+ hours of downtime due to two consecutive DDoS attacks targeted at the network interfaces on Amazon EBS (Elastic Block Store) service for storage used with EC2 instances. Such a problem could have been quickly resolved if the incidents were promptly diagnosed given sufficient visibility to the network traffic. The real issue is the multilevel administration of cloud services.

For BitBucket system administrators, the network traffic on the physical servers is a black box. Consequently, the administrators cannot do anything at the layers they cannot reach. The only solution is to rely on Amazon's support. However, after 6 hours, even with urgent request tickets and phone calls, Amazon's best advice was that EBS is a "shared network resource" and therefore performance would vary. Only afterward did Amazon acknowledge the problems with the service and work with BitBucket to resolve the problem.

It was determined that BitBucket was under a massive-scale DDoS attack using UDP packets. Amazon simply blocked the UDP traffic to resolve the problem. On the next day, another DDoS using TCP packets targeted BitBucket. But this time it took only two hours for Amazon and BitBucket to resolve the problem.

### D. *How can research help?*

Failure patterns similar to the S3 failure were observed in an error-injection based experimental analysis [2] of the Ensemble Group Communication System (GCS), a robust communication layer for distributed dependable applications. The study shows that about 5–6% of application failures are due to an error escaping the GCS error-containment mechanism and manifesting as silence data corruption. It is important to note that although the percentage of the observed silence data corruption is relatively small, such errors do constitute an impediment to achieving high dependability because recovery from these failures can involve significant system downtime.

Validation of such large-scale cloud computing deployments is always challenging, yet it is an interesting opportunity for the research community. An example of work in this direction is CloudVal [3], a fault injection framework designed to validate the dependability of cloud virtualization infrastructure. The paper presents a case study using a set of representative fault models, such as transient faults (system sensitivity to soft errors), guest misbehavior faults (possibility of error propagation from guest system to hypervisor), performance faults (potential of race conditions), and maintenance faults (turning on/off selected hardware resources, such as the CPU). The study uncovers several defects in the implementation of KVM and XEN hypervisors.

In the following sections, along with discussing the underlying cloud technologies, we examine some of the limitations of current technologies and the root causes of dependability problems. TABLE 1 summarizes our discussion.

## III. DEPENDABILITY OF MULTICSORE PROCESSORS

Recent advancements in manufacturing technology have led to development of multi-core processors and CPU performance enhancement techniques. The reduction in the size of individual gates allows putting more transistors into a single integrated circuit die. Partitioning the available transistors into multiple cores limits the power consumption and reduces heat dissipation only to those cores that are currently active. These architectures demand new reliability techniques to ensure correct operations.

TABLE 2: A TAXONOMY OF MULTICORE TECHNIQUES TO TOLERATE PERMANENT FAILURES

| Technique | Brief Description | Comments |
|---|---|---|
| **Core Disabling** | Core must be disabled due to permanent failure. | Cannot utilize the functional part of the core |
| **Architectural Core Salvaging** (e.g., [5]) | Every core is utilized as a fault-free core to execute the part of the ISA that it can execute without any fault. On an error, state is transferred to a core that has been predetermined to be able to execute the faulty instruction. | Utilizes partially damaged cores; introduces hardware complexity to salvage cores; state transferring is expensive |
| **Microarchitectural Core Salvaging** (e.g., Core Cannibalization [7]) | Cores are viewed at the granularity of pipeline stages in lending resources to other cores in the event of a failure in any stage of a core. This improves lifetime chip performance by enabling more cores to be functional. | More hardware complexity due to micro-architectural connections and redundancy; cannot cover all the instructions (according to analysis of nonreplicated instructions in [5]) |
| **Software-based Core Salvaging** (e.g., Core Virtualization [6]) | System-level software virtualizes a set of partially faulty cores to provide logically correct virtual cores for application execution. | High performance overhead due to instruction emulation in software |

### A. *Permanent failure tolerance in multicore processors*

Permanent (hardware) failures in processors have become more prevalent due to increases in core area and in the number of cores. Several high-level techniques have been proposed for tolerating such errors in multicore architectures. Besides the option to disable the faulty core, other advanced techniques, such as core salvaging, core virtualization, and core cannibalization, attempt to use working sections of a faulty core to improve overall chip performance. TABLE **2** summarizes and critiques the key concepts behind this work.

### B. *Transient error tolerance in multicore processors*

Smaller device sizes do reduce the probability of a strike from cosmic radiation on an individual device, they also reduce the charge required for a charged particle to flip the state and cause a single event upset. In addition, the increasing clock frequency of the processor increases the possibility of an upset in the combinational logic to be latched by a flip-flop. These trends in multicore necessitate transient error tolerance in these processors.

On the positive side, multicore architectures bring the possibility of supporting hardware redundancy in both a static and dynamic manner. TABLE **3** summarizes two common core-binding techniques for transient error tolerance in multicore architectures.

As an example of redundancy at the core level, IBM provided multiple levels of error detection and recovery for the G5 and G6 processors, starting from protecting the memory arrays on the chip and going to chip-level techniques in a multichip module (MCM). Multiple processor cores are interconnected to form an MCM. A special processor on the MCM is designated as the system processor. In case of a failure of one of the processors, its state is migrated to another idle processor in the MCM; this is called Transparent Processor Sparing. This approach requires an efficient mechanism to determine the state to be checkpointed and to capture that state periodically. Using multicore processors in the context of an application such as cloud computing would present the challenge of developing more sophisticated models for core-level checkpointing.

### C. *Challenges of using multicores in the cloud*

Multicore processors are the power behind cloud computing. Challenges in exploiting the potential redundancies offered by multi-core technologies include:

- Increasing levels of integration at the node level to produce large computing clusters prohibits (or makes it expensive) the straightforward application of replication techniques at the system level.
- Increasing CPU usage leads to higher error rates in memory [13]. More rigorous study is needed to evaluate how this issue affects the aging of cloud hardware.
- Most reliability techniques at the processor architecture level need specialized hardware, but cloud computing uses commodity multicore processors. Furthermore, many of these techniques are in early stages of descriptions and may have yet-to-be identified issues in implementation and, in assessing the real improvement in availability they bring for the added complexity.
- There is a need for a framework to effectively combine the tolerance techniques for hard and transient errors.

## IV. DEPENDABILITY OF VIRTUAL MACHINES

The use of virtual machine (VM) based systems introduces the hypervisor between the operating system and the hardware. The relationship between the hypervisor, also called the virtual machine monitor (VMM), and the guest operating system is analogous to the traditional relationship between the operating system and the application processes running on it.

TABLE 3: A TAXONOMY OF MULTICORE TECHNIQUES TO TOLERATE TRANSIENT ERRORS

| Technique | Brief Description | Comments |
|---|---|---|
| **Static Core Binding** | Cores are assigned as spares statically, and an entire core is disabled upon fault in the core | • Any permanent failure makes a pair of cores unavailable<br>• Two cores in the pair have to run at the speed of the slower core (difference in the core speeds could be due to process variation) |
| **Dynamic Core Binding [4]** | Spare cores are allocated dynamically based on the allowable network configuration, and an entire core is disabled upon a fault in the core | • Complexity on the core interconnected network<br>• Requires more sophisticated models for core-level checkpointing |

TABLE 4: CATEGORIES OF EXISTING MECHANISMS FOR VM CHECKPOINTING

| Mechanism | Brief Description | Comments |
|---|---|---|
| Stop-and-Save | Stops a VM completely, and saves its state to persistent storage | • Large system downtime<br>• Provided by all major VMM systems |
| Low-Freq (interval >1h) based on live migration (e.g., CEVM, VNsnap) | Creates a VM replica on a remote node via live migration, then the remote node writes the replica to disk | • Significant recomputation during recovery, as checkpoint frequency is low<br>• Large overhead (maintain full replicas for a protected VM) |
| High-Freq (interval 10~1000 ms) based on live migration (e.g., Remus) | Maintains a VM replica on a separate physical node via live migration, and fails-over upon a failure | • Large overhead while migrating latest updates to the remote node continuously (~50% overhead for 50ms checkpoint interval)<br>• Fail-stop assumption |
| High-Freq (interval 10~1000 ms) based on incremental checkpointing (e.g., VM-μCheckpoint) | Maintains high frequency (intervals <1s) incremental checkpoint of dirty pages in main-memory; recovers from the stored checkpoint in the same process context | • Small overhead (6.3% for SPEC06, 17.5% for Apache)<br>• Reduced likelihood of checkpoint corruption<br>• High recovery overhead when suffering latent faults (must recover from the disk-based checkpoint) |

### *A. Virtual machine based dependability techniques*

#### *1) Fault and failure detection*

VMs provide a software layer between OS and hardware and enables monitoring of the behavior of the guest system. Vigilant [14] is an initial effort that applies machine learning to detect VM failures based on the correlation of events generated by monitors at the hypervisor layer. Intrusion detection systems (IDS) are moving toward out-of-host implementations, in which the hypervisor layer becomes an attractive option [15].

Monitoring at the hypervisor layer enables failure/attack isolation and hence, independent reporting of the observed incidents from the outside, without the possibility of being corrupted/manipulated by the failing guest system.

#### *2) Recovery*

Virtualization encapsulates each complete guest system into a virtual machine and provides a convenient way to capture snapshots of the system state. Therefore, checkpoint and rollback are the primary recovery mechanisms in virtualization environment. TABLE **4** lists the existing mechanisms of VM checkpointing.

In the first category of VM checkpoint, the VM is stopped completely to save its state in persistent storage, and then the VM resumes. This approach incurs a large system downtime during the checkpoint.

In the second category (e.g., CEVM [8] and VNsnap [9]), VM live migration and copy-on-write are employed to create replica images of VMs with low overhead. Then the image is written to disk in the background or by the separate physical node. This disk-based VM checkpointing is not scalable, as it stresses the storage system when many VM checkpoints need to be written at the same time. In addition, the checkpoint is susceptible to corruption due the low-frequency updating.

The last two categories are the high-frequency VM checkpointing based on live migration (e.g., Remus [10]) and incremental checkpoint in main memory (e.g., VM-μCheckpoint [11]). These checkpoint schemes cannot tolerate latent errors, since the checkpoint might contain dormant faults. A hierarchical architecture combining high-frequency, incremental checkpointing and low-frequency, disk-based checkpointing is proposed in [11]. This approach reduces checkpointing corruption and performance overhead, but it still achieves latent error tolerance.

### *B. Challenges of using virtualization in the cloud*

Although there has been substantial progress in improving VM checkpointing, it is still challenging to *minimize checkpoint performance overhead, checkpoint corruption, and checkpoint inconsistency in a seamless way in the cloud infrastructure*.

The non-uniform, dynamic geographic distribution of the nodes in the current cloud-computing environment violates the assumptions of traditional distributed systems regarding communication overhead. Legacy techniques such as synchronous and asynchronous checkpointing already incur significant overhead and cannot be applied naïvely in the new scenario without investigation. Added to that are the high and nondeterministic costs that result from the dynamic nature of the distributed system.

Recent studies [3] have shown that *VMs introduce new failure modes* at the interface between the hypervisor, the hardware, and the operating system. The study [3] of hypervisor resiliency to errors revealed defects in the implementations of both KVM and XEN hypervisors. These defects can significantly affect the availability and maintainability of the cloud. For example, turning on CPUs/cores may cause hypervisor failure (e.g., crash of the KVM-hypervisor when turning on a CPU core), or memory corruptions in a single VM can cause the cloud management system to hang.

The major technical challenge in implementing monitoring and recovery mechanisms at the hypervisor level is *the semantic gap between the guest system and the low-level hypervisor*. The semantic gap prevents the hypervisor from interpreting the behavior of the guest system from the set of observable events and states at the hypervisor's point of view. To address this problem, *virtual machine introspection* techniques [12] have been introduced. The current techniques use the knowledge of the internal structure of the guest system to extract its behavior. For example, Linux kernels manage processes by maintaining a list of *task_struct* data structure, which can be used to obtain the list of the running processes in the guest system.

TABLE 5: EXAMPLES OF CLOUD STORAGE SYSTEMS

| System | Brief Description | Comments |
|---|---|---|
| **Amazon Dynamo** | A highly available key-value storage system that powers Amazon store. Dynamo's design concepts have influenced dozens of new database systems. | • Membership management is at scales of thousands of machines (recall the Amazon outage because of gossip messages).<br>• Resolving data inconsistency is the developers' responsibility. |
| **Google Datastore** | A schema-less object datastore with a query engine and atomic transactions. | • Legacy applications with complex requirements have to readjust their data representation to fit with Datastore's model. |

The fundamental limitation of this approach is its dependence on the invariants of the guest system, mostly of the guest operating system. Therefore, the implementation of this approach must be adjusted whenever the used invariants change. For example, the location and structure of the *task_struct* vary in different versions of the Linux kernel, so the introspection tools have to be customized accordingly to operate correctly. In addition, significant effort is often required to obtain the meaningful OS invariants. The developers and maintainers of these tools need to have a deep understanding of the internal implementation of the interested systems. Unfortunately, this is almost impossible in case of a closed-sourced OS like any version of Windows OS.

## V. DEPENDABILITY OF THE STORAGE SYSTEM

The storage system is an important component in cloud system stacks. Enterprise organizations expect high availability to be ensured by the storage system, particularly with respect to production data. To meet these requirements, storage systems become more complex, leading to potential reliability problems. In this section, we briefly present examples (summarized in TABLE **5**) of how storage systems in the commercial cloud overcome these challenges.

### A. Amazon Dynamo

Dynamo is a highly available key-value storage system for the Amazon online store. Dynamo employs the principle of *eventual consistency* (weak consistency) to achieve better availability. This section discusses the trade-off decision and techniques to achieve high availability.

- *Data partitioning using consistent hashing.* To distribute the workload and ensure consistency, Dynamo stores data in a logical hash-ring. A hash-ring consists of nodes, in which each node is responsible for storing a range of keys. When read or write requests arrive, a hash (key) is calculated to identify the appropriate node for accessing the associated value.
- *Achieving eventual consistency using replication and resolving data conflicts using a vector clock.* Each key/value pair is replicated on the replica nodes. Since the rings are logically formed, different nodes in different datacenters can be placed on one logical ring. Even when one datacenter is destroyed, Dynamo is still able to recover data from replicas in other datacenters. Using logical vector timestamping, clients are able to resolve potential conflicts in the replicas at read time.
- *Failure detection and membership management using Gossip messages.* To detect failure promptly, Dynamo uses a distributed message exchange protocol called Gossip. Each node maintains a view containing the health status of its neighbor nodes. When a failure occurs, the node directly connected to failed node knows first; it then *gossips* a failure event to other nodes in its view. Next, these nodes continue to propagate the failure event to other nodes in their views.

### B. Google Datastore for App Engine

Using master/slave replication, Google chose to prioritize performance and read consistency over availability by asynchronously replicating data from the master datacenter to the slave. However, this architecture performs poorly in some situations, such as:

- Unstable clusters might cause the master and its slave to be out of sync. In this case, the system must switch to read-only periods to synchronize the master and the slave. This consequently results in write-unavailability periods for the users.
- In the unplanned downtime, it is possible for the user to lose write operations that occur near this period.

These issues become more severe as the service grows. After two years from initial deployment, Google App Engine had experienced serious performance degradation and several long service outages.

To solve this problem, Google had implemented High Redundancy Datastore (HRD) service. The HRD is able to tolerate planned maintenance and is highly resilient to unplanned infrastructure issues. However, both Google and their customers have to pay for this level of availability. Similarly to Amazon's Dynamo, HRD sacrifices performance and consistency to achieve higher availability, as its write latency increases and read consistency decreases.

One of the key technologies in HRD is the Paxos consensus algorithm [17]**,** which enables HRD to synchronize data across datacenters in real time. The Paxos algorithm is used to reach the agreement on a single value in the presence of failures.

### C. Challenges of Storage Systems in the Cloud

Data storage systems must effectively manage increasing volumes of data, while striving to maintain the availability and consistency of the stored data. The challenges include:

*Achieving scalability*. Given rapidly growing volumes of data, storage systems must be able to scale to customer demand. While traditional relational database management systems (RDBMS) provide rich functionalities and maintain ACID (atomicity, consistency, isolation, durability) properties, it takes a lot of effort to scale these systems to the cloud environment. Recent systems, such as Amazon Dynamo and Google Datastore, trade the functionalities of traditional RDBMS for scalability and availability.

*Improving performance*. Recently, some storage systems have elected to use solid state drives or RAM memory to enhance performance [16]. To further improve performance, both the storage layer and the processing layer require more advanced optimization techniques.

*Developing and deploying failure detection on a large scale.* At many current clouds' scales, failure is the norm rather than the exception. Failure may happen at any layer, from physical storage devices, to file system drivers, to the network layer. As the Amazon S3 incident (in section II) exemplifies, it is challenging to effectively coordinate failure detectors to ensure coverage and timely detection. Furthermore, automating a part of the recovery process is required to reduce operational costs and system downtime.

## VI. Cloud Infrastructure and Service Assessment

Evaluation of cloud services is difficult from the customer's standpoint. Real-world behavior of cloud services may differ from the service level agreement. This section discusses metrics for benchmarking cloud services, highlighting key challenges of evaluating these metrics.

*Performance*. Performance metrics are usually stated in the service level agreement. Common performance metrics are response time, throughput, and others. One may measure these metrics by running a test using representative data. However, to simulate peak requests on a large scale, such as requests coming in from all over the world, is a big challenge to performance evaluation.

*Consistency.* In cloud, it is common practice to trade off consistency for availability. Due to the distributed nature of cloud computing, data are often replicated to data centers at different geographic locations. A proper consistency evaluation should initiate tests from different locations.

*Availability.* Availability is frequently defined in the service level agreement. Current cloud services provide a health dashboard to present status of services to customers. However, these dashboards' update intervals may not meet customers' needs in monitoring cloud services. Further, unavailable cloud services are rare, and significant time is required to quantify cloud availability.

*Fault Tolerance.* Since cloud systems are often built on top of commodity hardware, failure is a norm. A challenge is - how to evaluate the fault tolerance of the cloud in operational setting. It is difficult to accurately emulate physical hardware failures in cloud computing facilities or to simulate a large-scale disaster, such as a tornado or flood.

*Cost.* Choosing the right cloud provider and the right type of service to maximize return on investment is difficult. One may estimate the cost-effectiveness of different cloud services by projecting their application requirements when signing a service level agreement.

## VII. Conclusion

This paper attempted to critique succinctly the key techniques and techniques challenges building high availability into a cloud infrastructure, with focus on three enabling technologies: multi-core architecture, virtual machine, and the storage system.

## Acknowledgment

This work was supported in part by NSF grant CNS 10-18503 CISE, the Department of Energy under Award Number DE-OE0000097, the Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084, the Defense Threat Reduction Agency under award no. HDTRA1-11-1-0008, Boeing Corporation, and Infosys Ltd. The authors also thank Fran Baker for her careful reading of the draft of this paper.

## References

[1] "Amazon S3 Availability Event: July 20, 2008," http://status.aws.amazon.com/s3-20080720.html

[2] Basile, C. et al., "Group communication protocols under errors," In *Proc. of the 22nd Int'l Symposium on Reliable Distributed Systems*

[3] Pham, C. et al., "CloudVal: A framework for validation of virtualization environment in cloud infrastructure," *41st Annual Int'l Conf. on Dependable Systems Networks* (DSN '11)

[4] LaFrieda, C. et al., "Utilizing Dynamically Coupled Cores to Form a Resilient Chip Multiprocessor," In *Proc. of the 37th Annual Int'l Conf. on Dependable Systems and Networks* (DSN '07)

[5] Powell, M. et al., "Architectural core salvaging in a multi-core processor for hard-error tolerance," In *Proc. of the 36th Annual Int'l Symposium on Computer Architecture* (ISCA '09)

[6] Joseph, R. "Exploring salvage techniques for multi-core architectures," In *Workshop on High Performance Computing Reliability Issues* (HPCRI) 2005

[7] Romanescu, B. and Sorin, D., "Core cannibalization architecture: improving lifetime chip performance for multicore processors in the presence of hard faults," In *Proc. of the 17th Int'l Conf. on Parallel Architectures and Compilation Techniques* (PACT '08)

[8] Chanchio, K. et al., "An efficient virtual machine checkpointing mechanism for hypervisor-based HPC systems," in *Proc. of High Availability and Performance Computing Workshop*, 2008

[9] Kangarlou, A. et al., "VNsnap: taking snapshots of virtual networked environments with minimal downtime," in *Proc. of the 39th Annual Int'l Conf. on Dependable Systems and Networks* (DSN '09)

[10] Cully, B. et al., "Remus: high availability via asynchronous virtual machine replication," in *Proc. of USENIX Symposium on Networked Systems Design and Implementation*, 2008

[11] Wang, L. et al., "Checkpointing virtual machines against transient errors," *16th Int'l On-Line Testing Symposium* (IOLTS), 2010

[12] Payne, B.D. et al., "Secure and Flexible Monitoring of Virtual Machines," *23rd Annual Computer Security Applications Conf., 2007*

[13] Schroeder, B. et al., "DRAM errors in the wild: a large-scale field study," In *Proc. of the 11th ACM Int'l joint Conf. on Measurement and Modeling of Computer Systems* (SIGMETRICS '09)

[14] Pelleg, D. et al., "Vigilant: out-of-band detection of failures in virtual machines," *SIGOPS Oper. Syst. Rev.* 42, 1 (January 2008)

[15] Garfinkel, T. and Rosenblum, M., "A virtual machine introspection based architecture for intrusion detection," In *Proc. Net. and Distributed Sys. Sec. Symp.*, February 2003

[16] Ousterhout, J. et al., "The Case for RAMCloud," *Communications of the ACM*, Vol. 54, No. 7, July 2011

[17] L. Lamport, "The part-time parliament," ACM Trans. Comput. Syst., 16(2):133-169, 1998