SecondSite: Disaster Tolerance as a Service

Shriram Rajagopalan Brendan Cully Ryan O'Connor Andrew Warfield
Department of Computer Science, University of British Columbia
{rshriram, brendan, rjo, andy}@cs.ubc.ca

Abstract

This paper describes the design and implementation of SecondSite, a cloud-based service for disaster tolerance. SecondSite extends the Remus virtualization-based high availability system by allowing groups of virtual machines to be replicated across data centers over wide-area Internet links. The goal of the system is to commodify the property of availability, exposing it as a simple tick box when configuring a new virtual machine. To achieve this in the wide area, we have had to tackle the related issues of replication traffic bandwidth, reliable failure detection across geographic regions and traffic redirection over a wide-area network without compromising on transparency and consistency.

Categories and Subject Descriptors D.4.5 [*Operating Systems*]: Reliability—Backup procedures, Checkpoint/restart, Fault-tolerance

Keywords Wide Area Replication, Disaster Recovery

1. Introduction

Failures in the cloud are spectacular. In January 2010, Heroku, a Ruby-on-Rails application hosting company experienced the complete failure of their 22 virtual machines hosted on Amazon EC2. This outage, which was reportedly the result of a router failure, affected the 44,000 applications hosted on the site [7]. In May 2010, Amazon experienced four EC2 outages in a single week, the last of which resulted from a car crashing into a utility pole cutting power to a portion of the system [25]. Outages such as these are hardly limited to Amazon, who through the introduction of "availability zones", has taken steps to expose failure domains explicitly to customers so that applications may be designed to survive outages. However, while techniques to expose fate sharing and exposure to risk, such as Amazon's availability zones, help their customers engineer systems to survive failures, they are no panacea.

In April 2011, a "networking event" in the Virginia facility triggered a cascading failure in Amazon's Elastic Block Store (EBS), and resulted in a serious outage *across* multiple availability zones. An undisclosed number of virtual machines, including the hosts of a number of large, popular web-based applications, became unavailable for the initial twelve hours of the outage. Complete service was not restored for five days. Amazon published a detailed postmortem of the event, explaining that the outage stemmed from a series of bugs and unanticipated behaviors *in the failure recovery mechanisms* for EBS [37]. One of the concluding observations made in the document is that for the highest degree of availability, applications should be protected across multiple *regions*–geographically separate data center facilities that have a much lower degree of fate sharing than availability zones within the same data center.

The EBS failure is illustrative of the fact that while cloud computing environments are well maintained and professionally administered systems, they are not immune to outages. The postmortem, released less than a week after the outage itself, represents a surprisingly thorough and forthcoming explanation of the chain of events leading to failure. It also identifies a set of changes, to both software and administrative procedures, that will be taken to avoid repeating this sort of outage in the future. However, the failure severely impacted customers–even those who were making efforts to engineer their systems to be resistant to outages within the hosting environment–and left them without service for as many as five days. As such, the Virginia outage also demonstrates that while rare, outages are happening, and they are happening at scale. Even the skilled developers of mature Internet-based services face challenges in designing systems that handle these failures gracefully.

This paper describes ***SecondSite***, a high-availability and disaster tolerance service for virtual machines running in cloud environments. Based on the premise that implementing application- or OS-level fault tolerance is both difficult and expensive, we argue that facilities to enhance the availability of hosted software should be commodified, and offered as part of the hosting infrastructure. Just as spot markets [40] allow customers to receive a less reliable level of service at a cut rate, we argue that important workloads (and less sophisticated developers) would prefer a service that transparently provides a higher degree of availability. Using SecondSite, the owner of a hosted virtual machine may elect to have that VM's entire state continuously replicated to a backup image at an alternate geographic location. As with the Remus [11] system on which it is based, in the event of failure SecondSite allows the backup VM to take over in a completely transparent manner, requires no changes to OS or application code and exposes no failure to connected clients.

1.1 Failure Model

High-availability systems are typically complex: they require careful planning, complex integration with program logic, and ongoing maintenance as application code evolves over time. SecondSite makes two simplifying design decisions regarding its failure model that are intended to make it more practical for general purpose use.

Provide protection from fail-stop failures. Like Remus before it, SecondSite provides protection from fail-stop failures, the typical case in large-scale outages and disasters. The system provides transparent, continuous replication of a running VM to a passive backup host at another location. While fail-stop clearly includes destructive events such as fires and power outages, we also promote network failures to be fail-stop through the use of a watchdog that

VEE'12, March 3–4, 2012, London, England, UK.

	Local Area (Remus)	*Wide Area (SecondSite)*
Replication Link	Bonded Ethernet Interfaces	Routed IP connectivity over WAN
Replication Latency	Hundreds of microseconds	Tens of milliseconds
Failure Detection	Complete link failure of replication channel	Quorum among external vantage points
Network Relocation	Ethernet MAC (e.g. via ARP)	IP address (e.g. via BGP)

Table 1: Differences between the Local and Wide-Area Network.

automatically removes the active host from service in the case of network disconnection. This approach does not attempt to survive "wounded" applications experiencing partial failures as a result of overload, environmental problems, or bugs. However, it can help to maintain availability when a site is experiencing wide-area connectivity problems by transparently migrating to a better-connected backup during the network outage.

Protect the complete state of the running application. The failures for which SecondSite provides protection address concerns of *both* availability and durability. The established approach for disaster recovery in many environments today involves the asynchronous replication of storage. In the event of failure, storage-based disaster recovery systems (a) lose some amount of data and (b) require that the hosts reboot successfully from a crash-consistent image. In SecondSite, the disk image and complete running machine state are replicated, meaning that *no data* will be lost in the event of failure, and that reboots, with associated file system checks, are not required. This approach mitigates the risk associated with failures that may occur during restart on the backup host, such as problems checking the local (crash-consistent) file system.

In the next section, we characterize three challenges that we faced in evolving Remus to provide disaster tolerance as a service. The first of these concerns involve making replication work effectively over expensive and higher-latency wide-area links. The remaining two concerns address the network-related challenges of detecting failure and restoring service in an Internet, as opposed to local-area, environment. Many of the individual techniques described here have been used in other systems: the contribution of this work is to describe the development of an active disaster tolerant testbed that is currently deployed between two university data centers, approximately 260 kilometers apart. In addition to the architecture of the system, we describe several of the more painful corner cases that we experienced during development and deployment, and provide details of a continuous regression test suite that we use to validate and demonstrate confidence in the system to others.

2. Challenges of Wide-Area Disaster Tolerance

Remus [11] provides transparent high availability for unmodified OS and application software running within virtual machines on the Xen [4] hypervisor. Remus achieves this by employing checkpoint-based fault tolerance: during normal operation, incremental checkpoints of the entire VM state on the primary host are replicated to a backup host. These checkpoints are performed at a very high frequency (20-40 checkpoints/second) to enable fine grained recovery. On failure of the primary host, the VM at the backup host resumes execution from the latest completed checkpoint. The failover process is totally transparent to clients: the backup VM has the same IP address as the primary VM and, just as with live migration [9], the backup host issues a gratuitous ARP to ensure network packets going to the failed VM are automatically redirected to the backup VM as soon as it becomes available.

Remus checkpoints capture the entire state of the primary VM, which includes disk, memory, CPU and network device state. Checkpoint replication and acknowledgement is carefully pipelined and overlapped with execution to maintain low overhead, while still preserving consistency. The execution of the primary VM during each checkpoint interval is *speculative* until the interval has been checkpointed, since it will be lost if a failure occurs. Therefore, Remus uses output commit [29] to ensure that the external world sees a consistent view of the server's execution, despite failovers. More specifically, Remus queues and holds any outgoing network packets generated by the primary server until the completion of the next checkpoint, ensuring that unprotected speculative state is never exposed.

Output commit is also applied to isolate disk writes generated during a checkpoint interval. Writes received from the active server during an epoch are buffered and released to disk only after a checkpoint is complete. In the case of failure, Remus ensures that prior to resuming execution on the backup, all outstanding checkpointed writes have been written to disk and that any writes associated with speculative execution are discarded.

2.1 Challenges of Wide-Area Networks

Remus is designed for a LAN environment, and is intended to respond to the increased exposure to failure that results from consolidating many physical servers into virtual machines on a single host. In this environment, the failure of a single physical host carries elevated consequences, and Remus provided the ability to transparently recover from this class of failure by replicating state to a second physical server.

The environmental differences that face an availability system in moving from the local to the wide area are summarized in Table 1. First, the high bandwidth and low latency network of the local area is replaced with a typically lower bandwidth, and certainly higher latency IP connection between sites. As a consequence, the system must be more frugal with replication traffic, as networking resources are constrained and often also quite expensive.

A second difference is that Remus aims to survive any single, fail-stop physical failure. In Remus, replication traffic travelled over a dedicated point-to-point Ethernet link from the primary to the backup. This meant that Remus could survive the failure of any single physical component, even the loss of a replication NIC. In the wide area, the link between the active and backup hosts is often an IP route and may be lost even though the rest of the network remains available, requiring more careful consideration of failure notification. Finally, address relocation is considerably more challenging as a result of the need to alter upstream routing.

The remaining subsections discuss each of these three issues in more detail.

2.2 Bandwidth-Efficient Replication

Continuously replicating virtual machine checkpoints can require a lot of bandwidth. Depending on the frequency of checkpoint replication and the amount of state changed since the previous checkpoint, replication can easily consume up to 600 Mbps for memory intensive workloads. While a 1Gbps link supports this load, the cost of wide-area networking makes site-wide replication impractical at this rate. Furthermore, having multiple high-bandwidth streams

share a single link can increase congestion and reduce the effective bandwidth of the link.

In previous work on optimizing Remus for databases [24], we faced similar challenges: database workloads can generate very large checkpoints to the point that even fast LAN links may become performance bottlenecks. We developed three main techniques to reduce checkpoint size and latency: *Commit Protection* relaxes output buffering of some network messages. *Read tracking* avoids replicating changes to the contents of memory where that state has previously been read from disk. Finally, *Checkpoint Compression* saves bandwidth by performing online compression of the replication stream.

2.3 Failure Detection

To ensure correct execution, it is very important that only one of the replicas is active at once. When the replicas lose contact with each other, we must ensure that the active replica has failed (or become globally unreachable). Remus deals with the problem of distinguishing between system failure and network partition by bonding together two physical NICs for the replication link, so that it will continue to operate even if a single NIC fails. This does not suffice across a WAN, where the connection between the primary and the backup is not necessarily under the control of the high-availability service provider.

Differentiating between network partition and host failure using only two hosts is not easy, and we do not attempt to do it. Instead, we rely on an arbitration service residing outside of the protected network. If the primary and backup hosts lose contact with each other, they attempt to contact the arbitrator to decide which of them should continue running. They will terminate if instructed to do so by the arbitrator, or if they cannot reach the arbitrator.

2.4 Failure Recovery

When a virtual machine is relocated, network traffic between it and its clients must take a different path. The transition between locations must have two properties to be effective: it must be *comprehensive* — as the VM is being recovered in a geographically distant location, traffic from all clients must be appropriately redirected to the new VM instance. Second, in order to maintain availability, it must also complete in a *timely* manner. In the LAN environment, these properties are easy to achieve: As both primary and backup hosts are on a common Ethernet network, relocation is not exposed to the IP layer beyond the use of ARP.

Unlike the LAN environment, wide-area migrations involve BGP updates to redirect IP traffic from one site to another. BGP convergence times are challenging to reason about and slow BGP convergence may result in considerably longer periods of unreachability for the backup host. Rather than attempting to tackle the general issue of convergence delays related to IP address migration in BGP our approach uses a specific, but realistic BGP configuration to provide failover: we configure the protected network on the active and backup sites in the same manner that dual-homed IP networks are configured to survive link or ISP failures. Our network configuration can be thought of as a dual-homed set up in which not only is the backup link is redundant, but so are the servers that they connect to.

3. SecondSite Design

SecondSite builds upon Remus and provides the same transparency and consistency guarantees. In a typical deployment, SecondSite runs on each physical host on the primary site, replicating all virtual machines (VM) running on that host across a WAN to a host on the backup site. Each VM is checkpointed independently.

When the primary site fails, the backup site performs the required network reconfiguration (e.g., BGP updates) and resumes the execution of all the protected virtual machines from their last consistent checkpoint. The failure and subsequent migration to a remote location is completely transparent to the VMs.

3.1 Reducing Replication Overhead

RemusDB [24] introduced a set of optimizations to address the large overheads of checkpoint-based HA for database applications. These optimizations addressed both latency and replication bandwidth, and thus can be very beneficial for disaster recovery. The following were the main optimizations introduced by RemusDB:

- **Checkpoint Compression** Using *Page Delta* compression [30, 34] and an LRU cache of previously dirtied pages, this technique reduces the size of a checkpoint dramatically. The observation here was that updates to memory were sparse but spread over a large working set.
- **Disk Read Tracking** Since the disk is kept synchronized by Remus during replication, pages read from the disk by Guest VM need not be included in the checkpoint as long as it is unmodified during the checkpoint. At the backup host, these pages are periodically read from disk into the backup VM's memory.
- **Commit Protection** Applications can dynamically override output commit through a `setsockopt()` option. The ability to dynamically switch a connection between buffered and unbuffered states, allows transactional workloads like OLTP to leverage consistency guarantees of Remus without incurring the latency overhead introduced by output buffering [29].

We have only used one of these techniques (checkpoint compression) in this work. The read tracking optimization was developed against the Remus disk replication module, which does not support resynchronization after failure. Preliminary investigation indicated that it would not produce large improvements for the workloads we examined (discussed in Section 4), and so we have not yet ported support to our new disk synchronization layer.

Commit protection can reduce client-perceived latency dramatically, but it requires understanding of the protected application and cannot simply be switched on. As a platform-level service, SecondSite aims for transparency for both the protected VMs and the service provider, and so we have only evaluated techniques that conform to that goal.

3.2 Failure Detection in SecondSite

In SecondSite, we only consider *fail-stop* failures. A node may suspect failure if the replication stream times out, but the failure of the replication link does not prove that the remote node itself has failed, or even that it has become unreachable to its clients. Incorrectly classifying link failure as node failure can lead to a *Split-Brain* scenario, where both primary and backup become active at the same time. Failure detection in distributed systems is a richly researched topic, with many possible approaches but no single perfect solution. Possible approaches include unreliable failure detectors [8], minimal synchrony [12] and partial synchrony [15]. Several variants of unreliable failure detectors have been proposed, such as heartbeat [1, 2], adaptive timeouts [16], gossip-based detection [32], and so on. Such approaches assume partition tolerance, which SecondSite does not require.

SecondSite's failure detector requires the following properties:

- Quick detection — The failure detection logic contributes to the *service downtime*, as network packets from VMs are not released until a decision is made.
- No false positives — Incorrectly classifying a live site as dead leads to split brain situation and loss of data consistency.

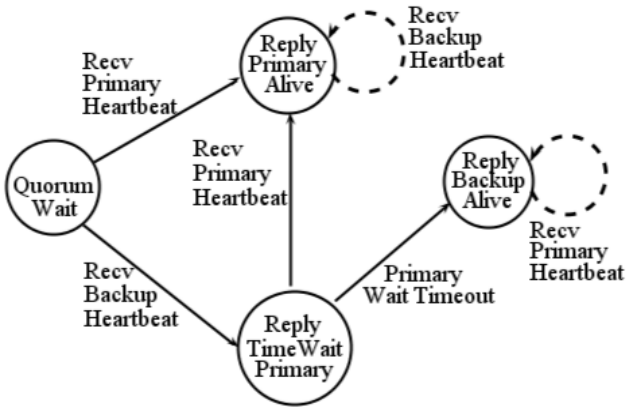


Figure 1: State Transitions of a Quorum Node

- Conservative failover — Network congestion can produce transient communication delays. Aggressively classifying such delays as failures and triggering recovery is expensive and could lead to reduced availability.

Quorum-based failure detection techniques can be used to achieve these requirements. There are several variants of the quorum technique. One simple technique is to use a *shared quorum disk*, where both nodes try to atomically reserve a disk partition (e.g. SCSI Reserve & Release). Such a solution is infeasible over a wide-area network. Another commonly used solution in WAN environments [41] is the *ping*-based approach. When the replication channel is broken, the backup site attempts to ping the primary site over a different network address. If the ping is successful, failover is avoided. This technique requires that there be at least two different routes between the primary and backup site, one to carry replication traffic and another for the ping test. This approach avoids requiring a third node by using distinct network interfaces to create a quorum. A third approach is to use quorum protocols [18] with one or more quorum nodes outside the WAN cluster acting as arbitrators.

Procedure 1 Quorum Logic at Primary Site

Require: $ReplicationTimeout == True$
  suspend VMs and stop replication
  result = SendHeartbeat(quorumNode)
  if result == PrimaryAlive **then** // *Link/Backup Failure*
    resume VMs
  else // *Failover happened*
    shutdown
  end if

Procedure 2 Quorum Logic at Backup Site

Require: $ReplicationTimeout == True$
  stop replication
  loop
    result = SendHeartbeat(quorumNode)
    if result == PrimaryAlive **then** // *Link Failure*
      shutdown
    else if result == TimeWaitPrimary **then**
      wait T secs for Primary to respond
    else if result == BackupAlive **then** // *Link/Primary failure*
      Issue BGP update to re-route network traffic
      resume VMs from latest checkpoint
    end if
  end loop

In SecondSite, we chose to use quorum server(s) to arbitrate during failure detection. We deployed a quorum web service in Google App Engine [39]. Figure 1 shows the state transitions of a quorum server used by SecondSite. The corresponding logic used by the primary and backup sites are illustrated in procedures 1 & 2 respectively. Using a quorum web service is just one example of how one could deploy a quorum service on the cloud cheaply. In fact, the quorum service we deployed consumed very little resources that it was well within the "daily free quota" offered by Google. When a failure is suspected by one node, a quorum is initiated. If the backup node initiates quorum first, it waits for a configurable time period for the primary node to make contact. The wait phase avoids unnecessary failovers caused by transient connectivity loss between the two nodes. While one quorum server is sufficient, to provide better redundancy one could deploy several quorum servers and use a simple majority quorum algorithm to decide which of the two nodes should be victimized in the event of network partition.

There are three failure scenarios to consider in our setup:

- **One Node Failure** Recovery succeeds with the help of quorum servers. If the surviving node is the backup node, failure recovery procedures are initiated.
- **Replication Link Failure** If a node obtains a quorum, it lives (or recovers). If not, it shuts down. It is thus possible that both nodes would shutdown if they fail to reach any quorum servers or the quorum nodes in any partition is insufficient to satisfy a majority. For services that cannot tolerate partitioned operation, this behavior ensures data consistency.
- **Quorum Node(s) Failure** Replication proceeds as usual.

Table 2 enumerates different failure scenarios, the quorum outcome and time to achieve the quorum (i.e. service downtime).

3.3 Failure Recovery

The dominant new challenge in recovering from failures in a wide-area environment is that of redirecting traffic: after failure detection determines that the backup site should step in, the IP addresses must be relocated as quickly as possible, between the two locations on the Internet. As Remus continues to perform output commit and the failure detection protocol ensures that at most one VM instance is ever active at a time, there is no possibility that open TCP sessions will ever move into an irrecoverable state due to mismatched sequence numbers. If IP advertisements change sufficiently quickly, failover will be almost completely transparent to clients; TCP sessions will remain open, and the only exposed aspect of failure will be the possibility of a small number of dropped packets during failover. To achieve this, SecondSite requires network support to quickly move IP addresses between the two sites.

Failing Component	*Quorum Outcome*	*Resolution Time (Service Downtime)*
Replication Link	Backup Shutdown	RTT to Quorum Service
Primary [+ Link]	failover to Backup	2 RTTs to Quorum Service + Quorum Timeout + Network Re-configuration
Backup [+ Link]	Primary survives	RTT to Quorum Service
Quorum Server	Replication continues	0
Primary + Quorum	Backup Shutdown	Service Failure (Manual Intervention)
Backup + Quorum	Primary Shutdown	Service Failure (Manual Intervention)
Link + Quorum	Primary & Backup Shutdown	Service Failure (Manual Intervention)

Table 2: Failure Scenarios and Down-times using Quorum Servers

The system achieves this by interacting with the Border Gateway Protocol (BGP) to influence Internet routing decisions.

BGP is a path vector protocol where every router selects the best route to destinations (IP address blocks) based on the routes advertised by neighboring routers. A BGP route advertisement for a given destination IP prefix contains the autonomous system numbers of all ASes it has traversed through, since originating from the AS responsible for that prefix. Route advertisements also have one or more attributes, that play a role in choosing the best possible route to a destination from a set of routes. Transitive attributes are passed on from one router to another, while non-transitive attributes are used to manipulate routing decisions between any two adjacent routers. When there are multiple routes to the same destination AS, the choice of the best route depends on the routing policies, AS path lengths and other attributes assigned to the route.

Figure 2 shows our routing architecture. We leverage BGP multi-homing to achieve failure recovery. On a high level, the system works as follows: the set of hosts being protected by SecondSite can be reached through two different Internet routes, one leading to the primary site and another to the backup site. Only the primary site's route is kept active (i.e. the best route) during normal operation. When a failover occurs, the backup site issues a new route advertisement that has preferable attributes to those of the existing preferred route. BGP route attribute manipulation enables us to influence the routing decisions made by upstream ASes.

An extensive measurement study done by Labovitz et al. [22] revealed that route updates from shorter to longer path lengths take longer time to converge compared to the converse case. As we are concerned with ensuring that reachability to the backup host be established as quickly as possible on failover, we have opted for a configuration in which a single upstream AS provides both links. We believe that this is realistic for many deployments, but also observe that the intuition from Labovitz et al. is that it is desirable to limit the number of ASes that advertisements need to propagate through to the greatest degree possible in order to reduce convergence times.

In practice, BGP multi-homing can be achieved in a number of ways. As examples, the following are some key attributes used to indicate route preferences:

- **Multiexit Discriminator (MED)**: When an AS has multiple entry points, the optional MED attribute can be used to provide a hint to neighbouring ASes about the preferred link for inbound traffic. As such, the MED attribute to switch inbound traffic from a primary to backup link. While MEDs are honored by current BGP implementations, they are non-transitive, i.e. a MED attribute received by an AS does not leave the AS. Thus, MED attribute based route manipulation only works in HA configurations that have a single upstream AS.
- **AS-path prepend**: AS-path prepending is a common way to influence BGP routing decisions of upstream routers. If there is no conflict with local routing policies, BGP routers generally select the route with the shortest path to a given destination prefix. Routers can be configured to indicate a reduced preference to the neighbouring ASes by advertising a longer AS-path for that prefix. AS-paths can be lengthened by prepending it one or more times with the router's AS number. AS-path prepending can be used as an HA mechanism for IP failover: The backup site advertises a longer path than the primary site for the IP prefix. On failover, it removes prepended entries to advertise a shorter path than the primary, resulting in a preferential route and so re-routing traffic to the backup site. This technique can be used even when there are multiple AS peerings at each site, however, the further upstream path adjustments have to travel in order to successfully redirect traffic, the more exposed the system will be to long convergence times and potential unreachability.
- **Communities and local preference**: A number of extensions have been added to BGP, largely with the intention of allowing ISPs to delegate a greater degree of control over routing decisions to their customers. BGP community and local preference attributes are two such extensions, and may be used in conjunction with AS-path prepending if the routing preferences should be visible beyond the ISP. The local preference attribute is used to influence routing decisions within an AS; routes with a higher local preference are favored over other routes. Communities allow collections of prefixes to be grouped and scoped together, and so act as a useful point of indirection to apply local preference updates.

  To influence routing decisions within the ISP, the client and the ISP agree to map certain BGP communities advertised by the client to local preferences within the ISP. This approach allows the client to use a private AS number, but will not propagate route changes beyond the ISP's AS without an analogous mapping of client BGP communities to ISP AS path prepends.

SecondSite uses BGP communities mapped to local preferences and AS path prepends, for failure recovery. The local preferences are used to influence routing decisions within our upstream AS. The community mappings to AS-path prepends influence routing decisions beyond our upstream AS. Both the primary and backup sites advertise a BGP route for a /24 CIDR block. All the protected VMs are assigned IP addresses from this block. The backup site's AS-path to the IP prefix is longer than the primary, as indicated in Figure 2, allowing the primary site to receive all network traffic for the VMs. When a failover is initiated, the backup site advertises a route with shorter AS-path and higher local preference, than the primary site. This causes traffic to flow to the backup site, once the routing update converges.

The approach that we have taken is hardly the only way to achieve route redirection with BGP. Other techniques, such as the use of more specific advertisements to redirect traffic are also possible. An earlier version of our system used this approach success-

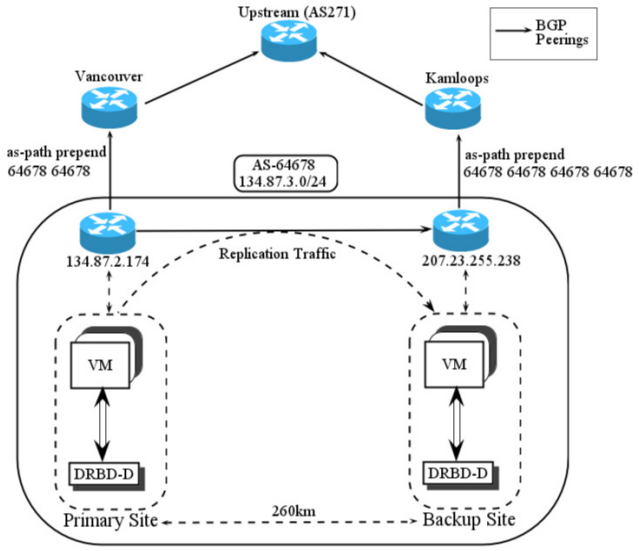


Figure 2: SecondSite setup over WAN

fully, but we were dissatisfied with the associated loss of IP address space that the technique required.

Further, BGP based IP multi-homing is not the only solution to transparently failover network connections. For example, in VPN based setups, routing reconfigurations could be done on the VPN concentrator [34], in order to maintain client connectivity during VM migration. Our concern with overlay network-based solutions such as these is that they often either introduced a single point of failure in the VPN concentrator, or led to situations in which traffic had to be routed through both active and passive sites during some classes of failure.

3.4 Seamless Failback

After a crashed primary site comes back online, in order to restart SecondSite, storage has to be resynchronized from backup to the primary site, without causing an outage to the VMs running in the backup site. Remus' storage replication driver, based on Xen's Blktap2 driver [36] does not provide a means for online resynchronization of storage. In order to understand the requirements of resynchronization, we turn the reader's attention towards the disk replication component of Remus.

Remus replicates disk writes asynchronously over the network to the backup site during a checkpoint interval. The backup site buffers in memory all writes received during a checkpoint. At the end of a checkpoint, primary site flushes any pending data in the socket and sends a *commit* message to the backup site. On reception of this message, the backup site sends an acknowledgement and then asynchronously flushes the buffered writes to disk. When the primary site fails during a checkpoint period, the backup site discards all buffered disk writes accumulated in that unfinished checkpoint.

A resynchronization module needs the following two functionalities:

- *Track blocks written by VMs at backup site after failover*. When the primary site is back online, these blocks have to be replicated from the backup to primary site, while the VMs continue performing i/o. This requires an online algorithm for dirty block tracking and resynchronization.
- *Discard writes made by primary site during the last unfinished checkpoint*. While it is easy for the backup site to compute these blocks from its checkpoint buffer and include them as part of dirty block resynchronization, it would be incomplete since replication is asynchronous. Only the primary site would have complete knowledge of blocks written before failure. This would necessitate the primary site to log the location of disk writes before doing the actual write. During resynchronization, the primary site would use the log to identify writes in the unfinished checkpoint prior to failure and overwrite them with data from the backup site.

DRBD [27] provides storage replication in asynchronous or synchronous mode with efficient *online resynchronization* functionality. Its *quick-sync* bitmap feature tracks writes done by a backup node after primary failure. Meanwhile, the primary node uses *activity logging* to keep track of active extents (4MB in size) during normal operation. It offers a variety of resynchronization options that suit SecondSite's needs. However, the current replication modes in DRBD do not support the checkpoint based asynchronous replication required by SecondSite. We modified DRBD to add a new Remus style replication protocol. DRBD is configured to perform a *one-way* resynchronization from backup to primary site, of all blocks indicated by the quick-sync bitmap at backup and activity log at primary.

During resynchronization period, the VMs at backup site continue to operate normally while the DRBD driver performs online storage resynchronization to bring the peer node up-to-date. Once the resynchronization is complete, SecondSite replication can be restarted very easily from either the primary site after live migrating the VMs back to it, or by reversing roles of primary and backup site.

4. Evaluation on a WAN Testbed

We have deployed SecondSite to provide continual protection for servers running in their home location at UBC in Vancouver, BC, backed up to a site 260 kilometers away at Thompson Rivers University in Kamloops, BC. The two sites are connected by a 1-Gigabit link with an average round trip time of 5ms. Figure 2 shows the topology of our configuration. The primary and backup servers are equipped with 8-core Intel Xeon processors and 16GB of RAM, and run Xen Linux 2.6.18-8 on Xen 4.0.0. SecondSite was configured to checkpoint the VMs on the primary host at 50ms epoch intervals, for a checkpoint frequency of 20 checkpoints per second.

4.1 Regression Tests

Using a high-latency routed link for replication applies stress to the checkpointing, failure detection and recovery systems that is not experienced over an Ethernet link. We felt the best way to validate our system was to run continuous regression tests under a real deployment between two distinct sites. The test suite consisted of a cluster of VMs (each with 512MB RAM and 1 vCPU) protected by SecondSite. The VMs ran synthetic workloads that stressed disk, memory and network subsystems, as described in Table 3. As they ran, we regularly triggered site failures. On failover, the test programs verified the integrity of their memory and files to ensure that data was not corrupted. When a crashed site came back online, its disks were resynchronized while online using DRBD. Once this phase completed, protection restarted with the recovered primary host acting as the new backup. The VMs continued to run their workload without interruption throughout this process.

Test Name	Workload	Test Goal
MemoryStress	Continuous malloc, dirty and free. Check integrity of allocated memory upon failover.	Trigger memory corruption bugs in compression logic.
PagetableStress	Parameterized fork bomb.	Trigger memory corruption bugs not caught by MemoryStress.
IOStress	Reads and writes on a large file in different I/O modes (buffered, direct).	Trigger bugs in disk replication logic.

Table 3: The Regression Test Suite

The regression test suite turned up some interesting corner cases:

Caching Pagetable Pages. Occasionally, the VM running the PagetableStress workload suffered fatal crashes in the guest kernel, pointing to random processes as the fault origin. Once it became evident that the crash was caused by memory corruption, we were able to trace the culprit to stale entries in the LRU page cache of the compression module. During live migration, a guest VM's pagetable pages are canonicalized before transmitting to target host, such that all references to the host's machine frames are replaced by their corresponding guest physical frames. On the receiving end, this process is reversed. The compression code only cached normal pages and transmitted pagetable pages as is, without updating the cache. Thus, when a normal page became a pagetable page, the cache ended up having a stale version of the page with the valid bit still set. When the page became a normal page again, a page delta would be taken against the wrong version of the page, resulting in memory corruption at the receiving end. Simply evicting the cache copy of the pagetable page fixed the issue.

Write after Write Race in Disk Replication. Write-heavy workloads sometimes experienced disk data corruption on failover. However, we could not reproduce the bug when we repeated the experiment several times on a different set of machines. We finally traced the problem to request re-ordering at the disk controller level. When the backup flushes a disk checkpoint, it simply queues up the writes to disk, merging the checkpoint flush with an ongoing one. When there are overlapping writes in adjacent checkpoint flushes, the disk does not guarantee the order of these writes. Inserting a *Disk I/O Barrier Request* at the end of every checkpoint flush fixed the issue.

4.2 Protecting a Cluster of VMs

Our goal in this section is to evaluate the cost and effectiveness of SecondSite in a production environment. To that end, we have provisioned an aggressive mix of macrobenchmark workloads on a single server and enable concurrent protection on all of them. Specifically, we have installed 2 web servers and 2 databases with different physical resource allocations onto a host with 8 physical CPUs. Domain-0 is configured with a vCPU for each physical CPU. Two VMs run the Apache web server with 1G of RAM and 2 vCPUs each, serving dynamic web pages. The *SPECweb 2005 Ecommerce benchmark* [43] is used to generate the web server workloads. The remaining two VMs run the MySQL Database Server, with 2G of RAM and 2 vCPUs each. The *DVD-Store OLTP benchmark* [42] is used to generate the database workloads. The VMs are assigned IP addresses from a */24* CIDR block that is protected by SecondSite using the BGP failover mechanism described earlier. Each workload has 100 concurrent clients accessing the services via their protected IP addresses. The benchmarks were run with 10 minutes of warm-up interval and 30 minutes of measurement. The OLTP benchmark's mean user think time was set to 10s. We present the benchmark results in terms of operations per minute, where the operation represents a database transaction for OLTP or an HTTP page request for SPECweb.

Figure 3 shows the throughput in terms of operations per minute observed by the clients over a 50 minute measurement period. A failure was injected at the primary site after 15 minutes of execution. With the quorum timeout set to 10 seconds, the backup site took 13 seconds to fully recover all the services and re-establish network connectivity.

Restarting HA. SecondSite takes advantage of DRBD to resynchronize storage with the recovering primary site, without interrupting the execution of VMs on the backup site. Once storage resynchronization completes, VM replication can be resumed from the backup site (now acting as primary) to the primary site (now acting as backup). If the primary site is the preferred home for the VMs, they may optionally be migrated back at this point.

After a 15 minute outage period, the primary host was brought online. Resynchronization of each VM's disk was initiated in parallel. Resynchronization took 52s to complete after which HA was restarted sequentially for each service, with the backup site now acting as the new primary. All VMs were fully protected after 88 seconds, with a constant time overhead of 20 seconds per VM to achieve steady state replication. The throughput increases after failover because the VMs are running in an unprotected mode, and therefore do not incur any checkpointing overhead. The drop in overall throughput starts during disk resynchronization period and settles back to pre-failure levels after all VMs are protected.

Disk Resynchronization. In order to restart HA for a VM, its disk has to be resynchronized with the recovered primary site. Our current recovery approach is to perform DRBD resynchronization of all VMs in parallel. This is a simple approach that is easy to validate and moves the system through recovery stages in lock step. However, it can produce contention on a shared physical disk that reduces the overall throughput of resynchronization. We have also experimented with resynchronizing VMs sequentially, so that each VM can be re-protected as soon as its individual synchronization is complete. This approach reduces the total resynchronization time and dramatically reduces the median unprotected window. In a test similar to the workload in Figure 3, the total resynchronization time is reduced from 37 seconds to 29 seconds, and median unprotected time falls from 23 seconds to 7 seconds.

4.3 DRBD Resynchronization Delays

Failback relies on DRBD to provide disk resynchronization, and cannot proceed until it completes. The time required for this process depends on the amount and locality of data written to disk while the VM is disconnected from its backup. To give an idea of the costs, Figure 4 presents the amount of data changed by the OLTP workload (a mixture of sequential and random I/O) as a function of the outage period, along with the time required to resynchronize the disk.

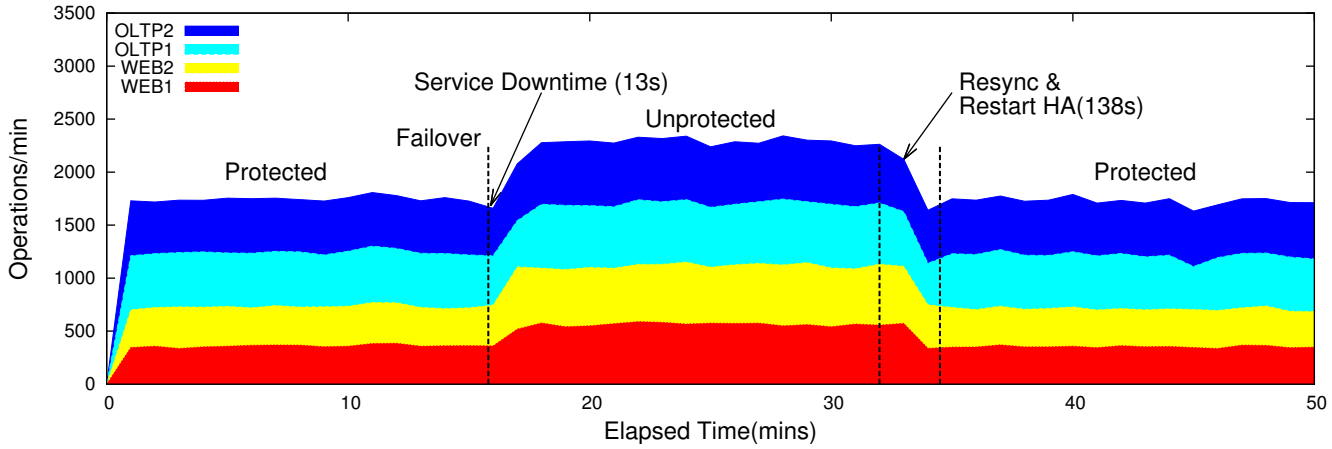


Figure 3: SecondSite Failover and Failback of a set of heterogeneous workloads.

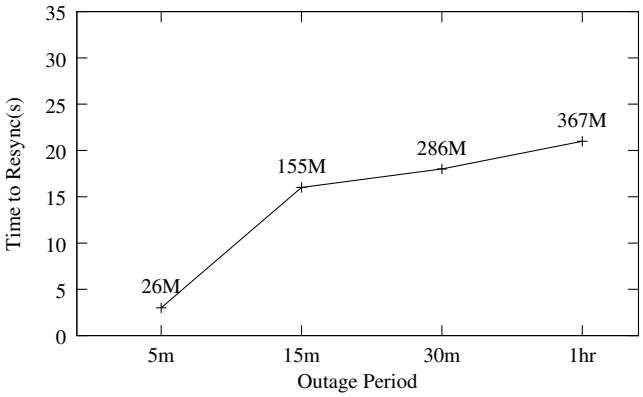


Figure 4: OLTP disk resynchronization costs

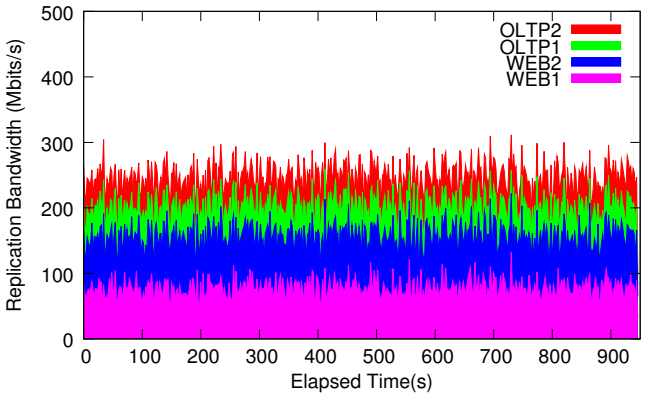


Figure 5: Replication Bandwidth Consumption before Failover (4 VMs x 100 Clients/VM)

Resource	Unprotected		Protected	
CPU Usage (Primary)	Dom0	VMs	Dom0	VMs
	1.1%	4.0%	12.0%	11.6%
CPU Usage (Backup)	–		Dom0	VMs
			8.4%	–
Replication Bandwidth	–		238.82 Mbps	

Table 4: Resource Consumption in a Multi-VM Environment

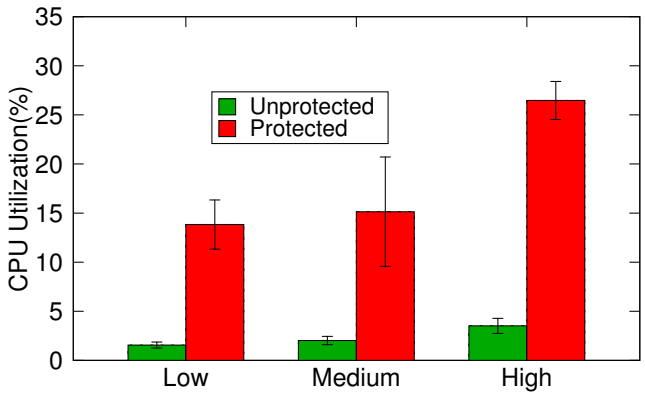
4.4 Resource Consumption & Throughput

Figure 5 shows the bandwidth consumption per VM when the entire site is protected by SecondSite. To get an idea of the CPU overhead incurred by SecondSite, we sampled the CPU utilization for all the domains on the physical host using the *xentop* [44] tool. All replication related overheads are accounted to Domain–0, while other VMs are executing. The CPU was sampled every 3 seconds over a one hour execution period. Xentop reports an additive percentage utilization over all physical cores. We normalize this value to 100% and report the CPU utilization of Domain–0 and aggregate CPU used by the four VMs in Table 4 for both unprotected and protected modes of operation. We also report the replication link bandwidth consumed by SecondSite during the one hour execution period in Table 4.

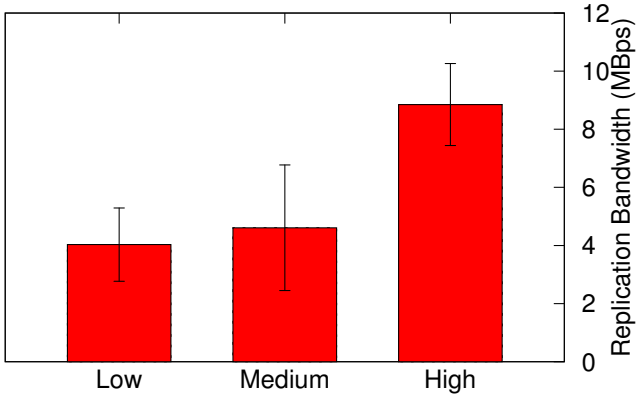
We also measure resources consumed by SecondSite (CPU and replication bandwidth) as a function of varying application loads. We vary the user's "think time" parameter of the OLTP workload, which in turn determines the number of requests/minute arriving at the server. We define three representative loads: High, Medium and Low, corresponding to think time values of 5s , 10s and 15s respectively. The aggregate CPU and network bandwidth consumed by Domain-0 are shown in Figure 6a and Figure 6b respectively.

4.5 Throughput vs Replication Latency

For a latency-sensitive workload like SPECweb, the throughput is limited by latency between the clients and the server. Our earlier work [11] examined the impact of network output buffering delay on SPECweb throughput, but did not examine the effects of replication link latency (there was none). In SecondSite, higher replication latencies result in longer times to commit the checkpoint at the backup site. Since the outgoing packets from the protected VM are buffered until the commit acknowledgement is received at the primary site, an increase in replication latency increases the protected server's response time. For workloads like SPECweb, this increased response time results in lower throughput, as illustrated in Figure 7.



(a) Domain-0's CPU Utilization



(b) Bandwidth usage on the ReplicationChannel

Figure 6: Cost of HA as a function of Application Load(OLTP Workload with 100 concurrent users).

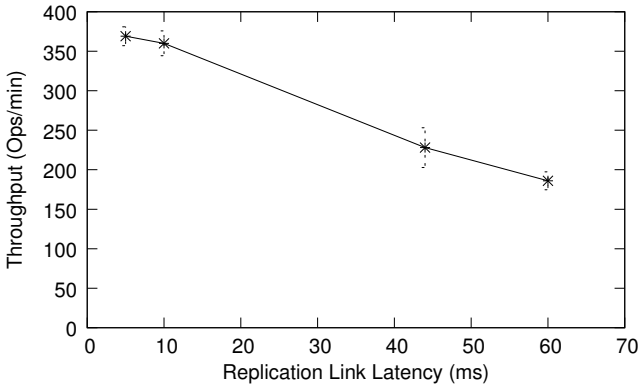


Figure 7: Impact of Replication Link Latency on Application Throughput (SPECweb with 100 Clients)

5. Related Work

5.1 Replication Overhead

Virtualization has been used to provide high availability for arbitrary applications running inside virtual machines, by replicating the entire virtual machine as it runs. Replication can be achieved either through event logging and execution replay or whole machine checkpointing. While event logging requires much less bandwidth than whole machine checkpointing, it is not guaranteed to be able to reproduce machine state unless execution can be made *deterministic*. Enforcing determinism on commodity hardware requires careful management of sources of non-determinism [6, 13], and becomes infeasibly expensive to enforce on shared-memory multiprocessor systems [3, 14, 35]. Respec [23] does provide deterministic execution recording and replay of multithreaded applications with good performance by lazily increasing the level of synchronization it enforces depending on whether it observes divergence during replay, but it requires intricate modifications to the operating system. It also requires re-execution to be performed on a different core of the same physical system, making it unsuitable for HA applications. For these reasons, the replay-based HA systems of which we are aware support only uniprocessor VMs [28]. SecondSite, like Remus [11] uses whole machine checkpointing, so it supports multiprocessor VMs.

Checkpoint-based replication builds on *live migration* [9] – the ability to migrate virtual machines from one physical host to another while they are running. The everRun VM [38] product, from Marathon Technologies, provides automated fault detection and failover for individual Windows VMs running over the Xen hypervisor. While it employs techniques similar to Remus, it is heavily tailored towards select Windows services like Exchange Server, SQL Server, etc. SecondSite on the other hand provides high availability through whole virtual machine replication with Xen [4] in an application and operating system agnostic manner, running over commodity hardware.

5.2 Minimizing Downtime

The established approach for disaster recovery in many environments today involves the synchronous or asynchronous replication of storage (e.g. SnapMirror [26], PipeCloud [33]). The Recovery Time Objective (RTO) and Recovery Point Objective (RPO) determines the degree of synchrony required and the overhead incurred during normal operation. Generally, some support is expected from the application level in order to restore the disk data to a consistent state.

Live migration has been successfully exploited by many systems to minimize *planned downtime*, when VMs are migrated across WAN. Storage migration over WAN has been explored by Bradford et. al [5] and Hirofuchi et. al [20]. Svärd et. al [30] evaluate the advantages of page delta compression for live migration for large enterprise class workloads.

CloudNet [34] demonstrated a viable cloud migration solution across data centers using storage migration, page delta compression and custom VPN concentrator components to maintain client connectivity during the migration. CloudNet primarily aims to tackle resource mobility and cloud burst scenarios for enterprise clouds. SecondSite could be integrated into CloudNet to achieve both dynamic resource pooling and fault tolerance across a WAN.

5.3 IP Migration across WAN

Harney et. al [19] use mobile IPv6 to solve the IP address migration issue. This system requires ISP network-level support for mobile IPv6 which is not yet widely available. Bradford et. al [5] propose the use of IP tunneling and dynamic DNS for maintaining network connectivity during live migration of a VM across WAN. Such a system is not resilient against site-wide failures or DNS caching at local proxies.

WOW [17] creates a VLAN on a P2P overlay network in which connectivity to the migrating virtual machine is maintained by forwarding it to the new destination. In WOW, the connectivity problem is restricted in scope to maintaining access only among peers of the P2P network. VIOLIN [21] uses a similar technique to create a VLAN on an overlay network, with software routers and switches to route traffic to appropriate nodes in the network. VIOLIN decouples the overlay virtual network from the underlying physical network, rendering changes in the underlying network topology transparent to the application. As with WOW, network transparency is only provided to participants in the overlay network

VM Turntable [31] explores the feasibility of long-haul live migration over WAN/MANs across two continents. Clients are required to establish IP Tunnels to the virtual machines and the network infrastructure at the destination takes care of reconfiguring the tunnel endpoint when the VM migrates across network domains. In contrast, SecondSite provides location transparency without requiring cooperation from clients.

6. Conclusion

Netflix weathered the April 2011 Amazon AWS outage with very little impact on its business [10]. Some key aspects of its cloud architecture include completely stateless services and NoSQL based data-stores that sacrifice consistency for availability and durability. SecondSite may not be a good choice for such stateless services. Building a scalable stateless e-commerce services like Netflix is *hard* and it requires an engineering skill that is generally not affordable by small and medium businesses. Application programmers use off-the-shelf solutions to rapidly develop and deploy web applications that often end up being stateful. We believe that SecondSite is a good fit for such applications as it eliminates the burden of maintaining consistency while adding high availability, transparent to the application.

Acknowledgments

We would like to thank Andree Toonk, Don McWilliam, Toby Wong, Michael Hrybyk and Marilyn Hay from BCnet and UBC IT for providing and supporting a WAN testbed for SecondSite. Andree in particular was invaluable in helping to explore how BGP could be configured to support failover. Thanks also to Cliff Harms and Brian Mackay at Thompson Rivers University for providing server hosting in Kamloops. Finally, we would like to thank the VEE reviewers for their insightful feedback and for accepting the paper for publication.

References

[1] M. K. Aguilera, W. Chen, and S. Toueg. Heartbeat: A timeout-free failure detector for quiescent reliable communication. Technical report, Ithaca, NY, USA, 1997.

[2] M. K. Aguilera, W. Chen, and S. Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. *Theor. Comput. Sci.*, 220:3–30, June 1999. ISSN 0304-3975.

[3] G. Altekar and I. Stoica. ODR: output-deterministic replay for multicore debugging. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 193–206, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-752-3.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-757-5.

[5] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pages 169–179, New York, NY, USA, 2007. ACM Press. ISBN 978-1-59593-630-1.

[6] T. C. Bressoud and F. B. Schneider. Hypervisor-based fault-tolerance. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles*, pages 1–11, December 1995.

[7] C. Brooks. Heroku learns the hard way from amazon ec2 outage. *SearchCloudComputing.com*, January 2010.

[8] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43:225–267, March 1996. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/226643.226647.

[9] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, Berkeley, CA, USA, 2005. USENIX Association.

[10] A. Cockroft, C. Hicks, and G. Orzell. Lessons Netflix Learned from the AWS Outage. http://techblog.netflix.com/2011/04/lessons-netflix-learned-from-aws-outage.html, April 2011.

[11] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: high availability via asynchronous virtual machine replication. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174, Berkeley, CA, USA, 2008. USENIX Association. ISBN 111-999-5555-22-1.

[12] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34:77–97, January 1987. ISSN 0004-5411.

[13] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 5th Symposium on Operating Systems Design & Implementation (OSDI 2002)*, 2002.

[14] G. W. Dunlap, D. G. Lucchetti, M. A. Fetterman, and P. M. Chen. Execution replay of multiprocessor virtual machines. In *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 121–130, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-796-4.

[15] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35:288–323, April 1988. ISSN 0004-5411.

[16] C. Fetzer, M. Raynal, and F. Tronel. An adaptive failure detection protocol. In *Proceedings of the 2001 Pacific Rim International Symposium on Dependable Computing*, PRDC '01, pages 146–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1414-6.

[17] A. Ganguly, A. Agrawal, P. Boykin, and R. Figueiredo. WOW: Self-Organizing Wide Area Overlay Networks of Virtual Workstations. *High-Performance Distributed Computing, International Symposium on*, 0:30–42, 2006.

[18] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the seventh ACM symposium on Operating systems principles*, SOSP '79, pages 150–162, New York, NY, USA, 1979. ACM. ISBN 0-89791-009-5.

[19] E. Harney, S. Goasguen, J. Martin, M. Murphy, and M. Westall. The efficacy of live virtual machine migrations over the internet. In *Proceedings of the 2nd international workshop on Virtualization technology in distributed computing*, VTDC '07, pages 8:1–8:7, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-897-8.

[20] T. Hirofuchi, H. Nakada, H. Ogawa, S. Itoh, and S. Sekiguchi. A live storage migration mechanism over wan and its performance evaluation. In *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, VTDC '09, pages 67–74, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-580-2.

[21] X. Jiang and D. Xu. VIOLIN: Virtual Internetworking on Overlay Infrastructure. In *ISPA*, pages 937–946, 2004.

[22] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed internet routing convergence. In *in Proc. ACM SIGCOMM*, pages 175–187, 2000.

[23] D. Lee, B. Wester, K. Veeraraghavan, S. Narayanasamy, P. M. Chen, and J. Flinn. Respec: efficient online multiprocessor replayvia speculation and external determinism. In *ASPLOS '10: Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, pages 77–90, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-839-1.

[24] U. F. Minhas, S. Rajagopalan, B. Cully, A. Aboulnaga, K. Salem, and A. Warfield. Remusdb: Transparent high availability for database systems. *PVLDB*, 4(11):738–748, 2011.

[25] C. C. T. A. P. Outage. R. miller. *datacenterknowledge.com*, May 2010.

[26] R. H. Patterson, S. Manley, M. Federwisch, D. Hitz, S. Kleiman, and S. Owara. SnapMirror: File-System-Based Asynchronous Mirroring for Disaster Recovery. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, page 9, Berkeley, CA, USA, 2002. USENIX Association.

[27] P. Reisner and L. Ellenberg. Drbd v8 – replicated storage with shared disk semantics. In *Proceedings of the 12th International Linux System Technology Conference*, October 2005.

[28] D. J. Scales, M. Nelson, and G. Venkitachalam. The design and evaluation of a practical system for fault-tolerant virtual machines. Technical Report VMWare-RT-2010-001, VMWare, Inc., Palo Alto, CA 94304, May 2010.

[29] R. Strom and S. Yemini. Optimistic recovery in distributed systems. *ACM Trans. Comput. Syst.*, 3(3), 1985.

[30] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, pages 111–120, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0687-4.

[31] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. de Laat, J. Mambretti, I. Monga, B. van Oudenaarde, S. Raghunath, and P. Y. Wang. Seamless live migration of virtual machines over the MAN/WAN. *Future Gener. Comput. Syst.*, 22:901–907, October 2006. ISSN 0167-739X. doi: 10.1016/j.future.2006.03.007.

[32] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Middleware '98, pages 55–70, London, UK, 1998. Springer-Verlag. ISBN 1-85233-088-0.

[33] T. Wood, H. A. Lagar-Cavilla, K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe. Pipecloud: using causality to overcome speed-of-light delays in cloud-based disaster recovery. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 17:1–17:13, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0976-9.

[34] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe. CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, pages 121–132, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0687-4.

[35] M. Xu, R. Bodik, and M. D. Hill. A "flight data recorder" for enabling full-system multiprocessor deterministic replay. *SIGARCH Comput. Archit. News*, 31(2):122–135, 2003. ISSN 0163-5964.

[36] Xen Blktap2 Driver. http://wiki.xensource.com/xenwiki/blktap2 .

[37] Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region. http://aws.amazon.com/message/65648/ .

[38] Marathon Technologies: everRun DR. http://www.marathontechnologies.com/ .

[39] Google app engine. http://code.google.com/appengine/ .

[40] Amazon EC2 Spot Instances. http://aws.amazon.com/ec2/spot-instances/ .

[41] VMware KB: Configuring Split-Brain Avoidance in a WAN. http://kb.vmware.com/kb/1008606 .

[42] Dell DVD Store Database Test Suite. http://www.delltechcenter.com/page/DVD+Store .

[43] SPECweb2005. http://www.spec.org/web2005/ .

[44] Xentop. http://linux.die.net/man/1/xentop .