Yank: Enabling Green Data Centers to Pull the Plug

Rahul Singh, David Irwin, Prashant Shenoy, and K.K. Ramakrishnan[‡]
University of Massachusetts Amherst [‡]AT&T Labs - Research

Abstract

Balancing a data center's reliability, cost, and carbon emissions is challenging. For instance, data centers designed for high availability require a continuous flow of power to keep servers powered on, and must limit their use of clean, but intermittent, renewable energy sources. In this paper, we present Yank, which uses a transient server abstraction to maintain server availability, while allowing data centers to "pull the plug" if power becomes unavailable. A transient server's defining characteristic is that it may terminate anytime after a brief advance warning period. Yank exploits the advance warning—on the order of a few seconds—to provide high availability cheaply and efficiently at large scales by enabling each backup server to maintain "live" memory and disk snapshots for many transient VMs. We implement Yank inside of Xen. Our experiments show that a backup server can concurrently support up to 15 transient VMs with minimal performance degradation with advance warnings as small as 10 seconds, even when VMs run memory-intensive interactive web applications.

1 Introduction

Despite continuing improvements in energy efficiency, data centers' demand for power continues to rise, increasing by an estimated 56% from 2005-2010 and accounting for 1.7-2.2% of electricity usage in the United States [16]. The rise in power usage has led data centers to experiment with the design of their power delivery infrastructure, including the use of renewable energy sources [3, 20, 21]. For instance, Apple's goal is to run its data centers off 100% renewable power; its newest data center includes a 40MW co-located solar farm [3].

Thus, determining the characteristics of the power infrastructure—its reliability, cost, and carbon emissions—has now become a key element of data center design [5]. Balancing these characteristics is challenging, since providing a reliable supply of power is often antithetical to minimizing costs (capital or operational) and emissions. A state-of-the-art power delivery infrastructure designed to ensure an uninterrupted flow of high quality power is expensive, possibly including i) connections to multiple power grids, ii) on-site backup generators, and iii) an array of universal power supplies (UPSs) that both condition grid power and guarantee enough time after an outage to spin-up and transition power to generators. Unfortunately, while renewable energy has no emissions, it is unreliable—generating power only intermittently based on uncontrollable environmental conditions—which limits its broad adoption in data centers designed for high reliability.

Prior research focuses on balancing a data center's reliability, cost, and carbon footprint by optimizing the power delivery infrastructure itself, while continuing to provide a highly reliable supply of power, e.g., using energy storage technologies [13, 14, 32] or designing flexible power switching or routing techniques [10, 24]. In contrast, we target a promising alternative approach: relaxing the requirement for a continuous power source and then designing high availability techniques to ensure software services remain available during unexpected power outages or shortages. We envision data centers with a heterogeneous power delivery infrastructure that includes a mix of servers connected to power supplies with different levels of reliability and cost. While some servers may continue to use a highly reliable, but expensive, infrastructure that includes connections to multiple electric grids, high-capacity UPSs, and backup generators, others may connect to only a single grid and use cheaper lower-capacity UPSs, and still others may rely solely on renewables with little or no UPS energy buffer. As we discuss in Section 2, permitting even slight reductions in the reliability of the power supply has the potential to decrease a data center's cost and carbon footprint.

To maintain server availability while also allowing data centers to "pull the plug" on servers if the level of available power suddenly changes, i.e., is no longer sufficient to power the active set of of servers, we introduce the abstraction of a *transient server*. A transient server's defining characteristic is that it may terminate anytime after an *advance warning* period. Transient servers arise in many scenarios. For example, spot instances in Amazon's Elastic Compute Cloud (EC2) terminate after a brief warning if the spot price ever exceeds the instance's bid price. As another example, UPSs built into racks provide some time after a power outage before servers completely lose power. In this paper, we apply the abstraction to green data centers that use renewables to power a fraction of servers, where the length of the warning period is a function of UPS capacity or expected future energy availability from renewables. In this context, we show that transient servers are cheaper and greener to operate than *stable servers*, which assume continuous power,

since they i) do not require an expensive power infrastructure that ensures 24x7 power availability and ii) can directly use intermittent renewable energy.

Unfortunately, transient servers expose applications to volatile changes in their server allotment, which degrade performance. Ideally, applications would always use as many transient servers as possible, but seamlessly transition to stable servers whenever transient servers become unavailable. To achieve this ideal, we propose system support for transient servers, called Yank, that maintains "live" backups of transient virtual machines' (VMs') memory and disk state on one or more stable *backup servers*. Yank extends the concept of whole system replication popularized by Remus [7] to exploit an advance warning period, enabling each backup server to support a many transient VMs. Highly multiplexing each backup server is critical to preserving transient servers' monetary and environmental benefits, allowing them to scale independently of the number of stable servers.

Importantly, the advance warning period eliminates the requirement that transient VMs always maintain external synchrony [23] with their backup to ensure correct operation, opening up the opportunity for both i) a looser form of synchrony and ii) multiple optimizations that increase performance and scalability. Our hypothesis is that a brief advance warning—on the order of a few seconds—enables Yank to provide high availability in the face of sudden changes in available power, cheaply and efficiently at large scales. In evaluating our hypothesis, we make the following contributions.

Transient Server Abstraction. We introduce the transient server abstraction, which supports a relaxed variant of external synchrony. Our variant, called just-in-time synchrony, exploits an advance warning that only ensures consistency with its backup before termination. We show how just-in-time synchrony applies to advance warnings with different durations, e.g., based on UPS capacity, and dynamics, e.g., based on intermittent renewables.

Performance Optimizations. We present multiple optimizations that further exploit the advance warning to scale the number of transient VMs each backup server supports without i) degrading VM performance during normal operation, ii) causing long downtimes when transient servers terminate, and iii) consuming excessive network resources. Our optimizations leverage basic insights about memory usage to minimize the in-memory state each backup server must write to stable storage.

Implementation and Evaluation. We implement Yank inside the Xen hypervisor and evaluate its performance and scalability in a range of scenarios, including with different size UPSs and using renewable energy sources. Our experiments demonstrate that a backup server can concurrently support up to 15 transient VMs with minimal performance degradation using an advance warning as small as 10 seconds, even when running memory-intensive interactive web applications, which is a challenging application for whole system replication.

Technique	*Overhead*	*Extra Cost*	*Warning*
Live Migration	None	None	Lengthy
Yank	**Low**	**Low**	**Modest**
High Availability	High	High	None

Table 1: Yank has less overhead and cost than high availability, but requires less warning than live migration.

2 Motivation and Background

We first define the transient server abstraction before discussing Yank's use of the abstraction in green data centers that use intermittent renewable power sources.

Transient Server Abstraction. We assume a virtualized data center where applications run inside VMs on one of two types of physical servers: (i) always-on *stable servers* with a highly reliable power source and (ii) *transient servers* that may terminate anytime. Central to our work is the notion of an *advance warning*, which signals that a transient server will shutdown after a delay T_{warn}.

Once a transient server receives an advance warning, a data center must move any VMs (and their associated state) hosted on the transient server to a stable server to maintain their availability. Depending on T_{warn}'s duration, two solutions exist to transition a VM to a stable server. If T_{warn} is large, it may be possible to live migrate a VM from a transient to a stable server. VM migration requires copying the memory and disk (if necessary) state [6], so the approach is only feasible if T_{warn} is long enough to accommodate the transfer. Completion times for migration are dependent on a VM's memory and disk size, with prior work reporting times up to one minute for VMs with only 1GB memory and no disk state [1, 19].

An alternative approach is to employ a *high availability mechanism*, such as Remus [7, 22], which requires maintaining a live backup copy of each transient VM on a stable server. In this case, a VM transparently fails over to the stable server whenever its transient server terminates. While the approach supports warning times of zero, it requires the high runtime overhead of continuously propagating state changes from the VM to its backup. In some cases, memory-intensive, interactive workloads may experience 5X degradation in latency [7]. Supporting an advance warning of zero also imposes a high cost, requiring a backup server to keep VM memory snapshots resident in its own memory. In essence, supporting a warning time of zero requires a 1:1 ratio between transient and backup servers. Unfortunately, storing memory snapshots on disk is not an option, since it would further degrade performance by reducing the memory bandwidth ($\sim$3000MB/s) of primary VMs to the disk bandwidth ($<$100MB/s) of the backup server.

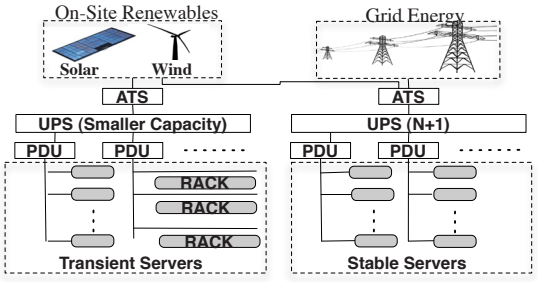


Figure 1: A data center with transient servers powered by renewables and low-capacity UPSs, and stable servers powered by the grid and redundant, high-capacity UPSs.

Live migration and high availability represent two extreme points in the design space for handling transient servers. The former has low overhead but requires long warning times, while the latter has high overhead but handles warning times of zero. Yank's goal is to exploit the middle ground between these two extremes, as outlined in Table 1, when a short advance warning is insufficient to complete a live migration, but does not necessarily warrant the overhead of externally synchronous live backups of VM memory and disk state. Yank optimizes for modest advance warnings by maintaining a backup copy, similar to Remus, of each transient VM's memory and disk state on a stable backup server. However, Yank focuses on keeping costs low, by highly multiplexing each backup server across many transient VMs.

As we discuss, our approach requires storing portions of each VM's memory backup on stable storage. We show that for advance warnings of a few seconds, Yank provides similar failover properties as high availability, but with an overhead and cost closer to live migration. In fact, Yank devolves to high availability for a warning time of zero, and reduces to a simple live migration as the warning time becomes larger. Yank focuses on scenarios where there is an advance warning of a fail-stop failure. Many of these failures stem from power outages where energy storage provides the requisite warning.

Green Data Center Model. Figure 1 depicts the data center infrastructure we assume in our work. As shown, the data center powers servers using two sources: (i) on-site renewables, such as solar and wind energy, and (ii) the electric grid. Recent work proposes a similar architecture for integrating renewables into data centers [18].

We assume that the on-site renewable sources power a significant fraction of the data center's servers. However, since renewable generation varies based on environmental conditions, this fraction also varies over time. While UPSs are able to absorb short-term fluctuations in renewable generation, e.g, over time-scales of seconds to minutes caused by a passing cloud or a temporary drop in wind speed, long-term fluctuations require switching servers to grid power or temporarily deactivating them. A key assumption in our work is that it is feasible to switch some, *but not all*, servers from renewable sources to the grid to account for these power shortfalls.

The constraint of powering some, but not all, servers from the grid arises if a data center limits its peak power usage to reduce its electricity bill. Since peak power disproportionately affects the electric grid's capital and operational costs, utilities routinely impose a surcharge on large industrial power consumers based solely on their peak demand, e.g., the maximum average power consumed over a 30 minute rolling window [10, 13]. Thus, data centers can reduce their electricity bills by capping grid power draw. In addition, to scale renewable deployments, data centers will increasingly need to handle their power variations locally, e.g., by activating and deactivating servers, since i) relying on the grid to absorb variations is challenging if renewables contribute a large fraction ($\sim$20%) of grid power and ii) absorbing variations entirely using UPS energy storage is expensive [13]. In the former case, rapid variations from renewables could cause grid instability, since generators may not be agile enough to balance electricity's supply and demand.

Thus, in our model, green data centers employ both stable and transient servers. Both server types require UPSs to handle short-term power fluctuations. However, we expect transient servers to require only a few seconds of expensive UPS capacity to absorb short-term power fluctuations, while stable servers may require tens of minutes of capacity to permit time to spin-up and transition to backup generators in case of a power outage.

3 Yank Design

Since Yank targets modest-sized advance warnings on the order of a few seconds, it cannot simply migrate a transient VM to a stable server after receiving a warning. To support shorter warning times, one option is to maintain backup copies (or snapshots) of a VM's memory and disk state on a dedicated stable server, and then continuously update the copies as they change. In this case, if a transient VM terminates, this *backup server* can restart the VM using the latest memory and disk snapshot. A high availability technique must commit changes to a VM's memory and disk state to a backup server frequently enough to support a warning time of zero. However, supporting a warning time of zero necessitates a 1:1 ratio between transient and backup servers, eliminating transient servers' monetary and environmental benefits.

In contrast, Yank leverages the advance warning time to scale the number of transient servers independently of the number of backup servers by controlling when and how frequently transient VMs commit state updates to the backup server. In essence, the warning time T_{warn} limits the amount of data a VM can commit to its backup server after receiving a warning. Thus, during normal op-

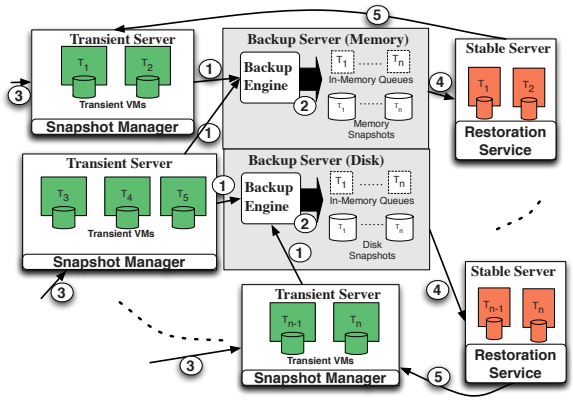


Figure 2: Yank's Design and Basic Operation

eration, a transient VM need only ensure the size of dirty memory pages and disk blocks remains below this limit. Maintaining this invariant guarantees that no update will be lost if a VM terminates after a warning, while providing additional flexibility over when to commit state. To keep its overhead and cost low, Yank highly multiplexes backup servers, allowing each to support many (>10) transient VMs by i) storing VM memory and disk snapshots, in part, on stable storage and ii) using multiple optimizations to prevent saturating disk bandwidth. Thus, given an advance warning, Yank supports the same failure properties as high availability, but uses fewer resources, e.g., hardware or power, for backup servers.

3.1 Yank Architecture

Figure 2 depicts Yank's architecture, which includes a *snapshot manager* on each transient server, a *backup engine* on each stable backup server, and a *restoration service* on each stable (non-backup) server. We focus primarily on how Yank maintains memory snapshots at the backup server, since we assume each backup server cannot keep memory snapshots for all transient VMs resident in its own memory. Thus, Yank must mask the order-of-magnitude difference between a transient VM's memory bandwidth (∼3000MB/s) and the backup server's disk bandwidth (< 100MB/s). By contrast, maintaining disk snapshots poses less of a performance concern, since the speed of a transient VM's disk and its backup server's disk are similar in magnitude. This characteristic combined with a multi-second warning time permits asynchronous disk mirroring to a backup server without significant performance degradation. Thus, while many of our optimizations below apply directly to disk snapshots, Yank currently uses off-the-shelf software (DRBD [9]) for disk mirroring.

Figure 2 also details Yank's functions. The snapshot manager executes within the hypervisor of each transient server and tracks the dirty memory pages of its resident VMs, periodically transmitting these pages to the backup engine, running at the backup server (1). The backup engine then queues each VM's dirty memory pages in its own memory before writing them to disk (2). Yank includes a service that monitors UPS state-of-charge, via the voltage level across its diodes, and translates the readings into a warning time based on the power consumption of transient servers (3). The service both i) informs backup and transient servers when the warning time changes and ii) issues warnings to transient and backup servers of an impending termination due to a power shortage. Since Yank depends on warning time estimates, the service above runs on a stable server. We discuss warning time estimation further in Section 3.5.

Upon receiving a warning (3), the snapshot manager pauses its VMs and commits any dirty pages to the backup engine before the transient server terminates. The backup engine then has two options, assuming it is too resource-constrained to run VMs itself: either store the VMs' memory images on disk for later use, or migrate the VMs to another stable (non-backup) server (4). Yank executes a simple restoration service on each stable (non-backup) server to facilitate rapid VM migration and restoration after a transient server terminates.

3.2 Just-in-Time Synchrony

Since Yank receives an advance warning of time T_{warn} before a transient server terminates, its VM memory snapshots stored on the backup server need not maintain strict external synchrony [23]. Instead, upon receiving a warning of impending termination, Yank only has to ensure what we call *just-in-time synchrony*: a transient VM and its memory snapshot on the backup server are always capable of being brought to a consistent state before termination. To guarantee just-in-time synchrony, as with external synchrony, the snapshot manager tracks dirty memory pages and transmits them to the backup engine, which then acknowledges their receipt. However, unlike external synchrony, just-in-time synchrony only *requires* the snapshot manager to buffer a VM's externally visible, e.g., network or disk, output when the size of the dirty memory pages exceed an upper threshold U_t, such that it is impossible to commit any more dirty pages to the backup engine within time T_{warn}.

In the worst case, with a VM that dirties pages faster than the backup engine is able to commit them, the snapshot manager is continually at the threshold, and Yank reverts to high availability-like behavior by always delaying the VM's externally visible output until its memory snapshot is consistent. Since we assume the backup server is not able to keep every transient VM memory snapshot resident in its own memory, the speed of the backup engine's disk limits the rate it is able to commit page changes. While memory bandwidth is an order of magnitude (or more) greater than disk (or network) bandwidth, Yank benefits from well-known characteristics of

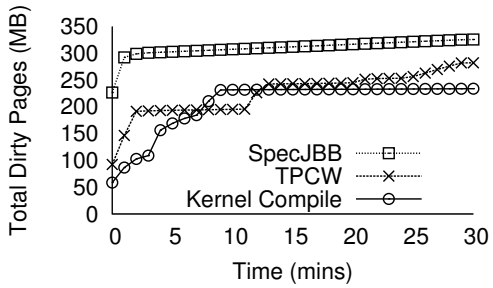


Figure 3: Working set size over time for three benchmarks: SPECjbb, TPCW, and Linux kernel compile.

typical in-memory application working sets to prevent saturating disk (or network) bandwidth. Specifically, the size of in-memory working sets tend to i) grow slowly over time and ii) be smaller than the total memory [8].

Slow growth stems from applications frequently re-writing the same memory pages, rather than always writing new ones. Yank only has to commit the last re-write of a dirty page (and not the intervening writes) to the backup server after reaching its upper threshold of dirty pages U_t. In contrast, to support termination with no advance warning, a VM must commit nearly *every* memory write to the backup server. In addition, small working sets enable the backup engine to keep most VMs' working sets in memory, reducing the likelihood of saturating disk bandwidth from writing dirty memory pages to disk. Recent work extends this insight to collections of VMs in data centers, showing that while the size of a single VM's working set may experience temporary bursts in memory usage, the bursts are often brief and not synchronized across VMs [33]. Yank relies on these observations in practice to highly multiplex each backup server's memory without saturating disk bandwidth or degrading transient VM performance during normal operation.

To confirm the characteristics above, we conducted a simple experiment: Figure 3 plots the dirty memory pages measured every 100ms for a VM with 1GB memory over a thirty minute period with three different applications: 1) the SPECjbb benchmark with 400 warehouses, 2) the TPC-W benchmark with 100 clients performing a browsing workload and 3) a Linux kernel (v 3.4) compile. The experiment verifies the observations above, namely that in each case i) the VM dirties less than 350MB or 35% of the available memory and ii) after experiencing an initial burst in memory usage the working set grows slowly over time.

Yank also relies on the observations above in setting its upper threshold U_t. To preserve just-in-time synchrony, this threshold represents the maximum size of the dirty pages the snapshot manager is capable of committing to the backup engine within the warning time. Our premise is that as long as the backup engine is able to keep each VM's working set in memory, even if all VMs simultaneously terminate, it should be able to commit any outstanding dirty pages without saturating disk bandwidth. Thus, U_t is a function of the warning time, the available network bandwidth between transient and backup servers, and the number of transient servers that may simultaneously terminate. For instance, for a single VM hosted on a transient server using a 1Gbps network link, a one second advance warning results in U_t=125MB. In Figure 3 above, U_t=125MB would only force SpecJBB to pause briefly on startup (where its usage briefly bursts above 125MB/s). The other applications never allocate more than 125MB in one second.

Of course, to support multiple VMs terminating simultaneously requires a lower U_t. However, as discussed in Section 3.5, Yank bases its warning time estimates on an "empty" UPS being at 40-50% depth-of-discharge. Thus, U_t need not be exactly precise, providing time to handle unlikely events, such as a warning coinciding with a correlated burst in VM memory usage, which may slow down commits by saturating the backup engine's disk bandwidth, or all VMs simultaneously terminating.

3.3 Optimizing VM Performance

For correctness, just-in-time synchrony only requires pausing a transient VM and committing dirty pages to the backup engine once their size reaches the upper threshold U_t, described above. A naïve approach only employs a single threshold at U_t by simply committing all dirty pages once their size reaches U_t. However, this approach has two drawbacks. First, it forces the VM to inevitably delay the release of externally visible output each time it reaches U_t, effectively pausing the VM from the perspective of external clients. If the warning time is long, e.g., 5-10 seconds, then the U_t will be large, causing long pauses. Second, it causes bursts in network traffic that affect other network applications.

To address these issues, the snapshot manager also uses a lower threshold L_t. Once the size of the dirty pages reaches L_t, it begins asynchronously committing dirty pages to the backup engine until the size is less than L_t. Algorithm 1 shows pseudo-code for the snapshot manager. The downside to using L_t is that the snapshot manager may end up committing more pages than with a single threshold, since it may unnecessarily commit the same memory page multiple times. Yank uses two techniques to mitigate this problem. First, to determine the order to commit pages, the snapshot manager associates a timestamp with each dirty page and uses a Least Recently Used (LRU) policy to prevent committing pages that are being frequently re-written. Second, the snapshot manager adaptively sets the lower threshold to be just higher than the VM's working set, since the working set contains the pages a VM is actively writing.

As we discuss in Section 5, our results show that as

Algorithm 1: Snapshot Manager's Algorithm for Committing Dirty Pages to the Backup Engine

```
Initialize Dirty Page Bitmap;
while No Warning Signal do
    Check Warning Time Estimate;
    Convert Warning Time to Upper Threshold (U_t);
    Get Num. Dirty Pages;
    if Num. Dirty Pages > U_t then
        Buffer Network Output of Transient VM;
        Transmit Dirty Pages to Backup Engine to
          Reduce Num. Dirty Pages to Lower Threshold (L_t);
        Wait for Ack. from Backup Engine;
        Unset Dirty Bitmap for Pages Sent;
        Release Buffered Network Packets;
    end
    if Num. Dirty Pages > L_t then
        Transmit Dirty Pages to Backup Engine at Specified Rate;
        Wait for Ack. from Backup Engine;
        Unset Dirty Bitmap for Pages Sent;
    end
end
Warning Received;
Pause the Transient VM;
Transmit Dirty Pages to Receiver Service;
Destroy the Transient VM;
```

long as the size of the VM's working set is less than U_t, using L_t results in smoother network traffic and fewer, shorter VM pauses. Of course, a combination of a large working set and short warning time may force Yank to degrade performance by continuously pausing the VM to commit frequently changing memory pages. In Section 5, we evaluate transient VM performance for a variety of applications with advance warnings in the range of 5-10 seconds. Finally, the snapshot manager implements standard optimizations to reduce network traffic, including content-based redundancy elimination and page deltas [34, 35]. The former technique associates memory pages with a hash based on their content, allowing the snapshot manager to send a 32b hash index rather than a 4kB memory page if the page is already present on the backup server. The technique is most useful in reducing the overhead of committing zero pages. The latter technique allows the snapshot manager to only send a page delta if it has previously committed a memory page.

3.4 Multiplexing the Backup Server

To multiplex many transient VMs, the backup engine must balance two competing objectives: using disk bandwidth efficiently during normal operation, while minimizing transient VM downtime in the event of a warning.

3.4.1 Maximizing Disk Efficiency

The backup engine maintains an in-memory queue for each transient VM to store newly committed (and acknowledged) memory pages. Since the backup server's memory is not large enough to store a complete memory snapshot for every transient VM it supports, it must inevitably write pages to disk. Yank includes multiple optimizations to prevent unnecessary disk writes.

First, when a transient VM commits a change to a memory page already present in its queue, the receiver deletes the out-of-date memory page without writing it to disk. As a result, the backup engine can often eliminate disk writes for frequently changing pages, even if the snapshot manager commits them. Second, to further prevent unnecessary writes, the backup engine orders each VM's queue using an LRU policy. Our use of LRU in both the snapshot manager (when determining which pages to commit on the transient VM) and the backup engine (when determining which pages to write to disk on the backup server) follows the same principles as a standard hierarchical cache. In addition, to exploit the observation that bursts in VM memory usage are not highly correlated, the backup engine selects pages to write to disk by applying LRU globally across all VM queues. Since it allocates a fixed amount of memory for all queues (near the backup server's total memory), the global LRU policy allows each VM's queue to grow and shrink as its working set size changes.

Finally, to further maximize disk efficiency, the backup engine could also use a log-structured file system [25], since its workload is write mostly, read rarely (only in the event of a failure). We discuss this design alternative and its implications in the next section.

3.4.2 Minimizing Downtime

The primary bottleneck in restoring a transient VM after a failure is the time required to read its memory state from the backup server's disk. Thus, to minimize downtime, as soon as a transient VM receives a warning, the backup engine immediately halts writing dirty pages to disk and begins reading the VM's existing (out-of-date) memory snapshot from disk, without synchronizing it with the queue of dirty page updates. Instead, the backup engine first sends the VM memory snapshot from disk to a restoration service, running on the destination stable (non-backup) server. We assume that an exogenous policy exists to select a destination stable server in advance to run a transient VM if its server terminates. In parallel, the backup engine also sends the VM's in-memory queue to the restoration service, after updating it with any outstanding dirty pages from the halted transient VM. To restore the VM, the restoration service applies the updates from the in-memory queue to the memory snapshot from the backup server's disk without writing to its own disk. Importantly, the sequence only requires reading a VM's memory snapshot from disk once, which begins as soon as the backup engine receives the warning. Note that if multiple transient VMs fail simultaneously, the backup engine reads and transmits memory snapshots from disk one at a time to maximize disk efficiency and minimize downtime across all VMs.

There are two design alternatives for determining how

the backup engine writes dirty page updates to its disk. As mentioned above, one approach is to use a log-structured file system. While a pure log-structured file system works well during normal operation, it results in random-access reads of a VM's memory snapshot stored on disk during failover, significantly increasing downtime (by ∼100X, the difference between random-access and sequential disk read bandwidth). In addition, maintaining log-structured files may lead to large log files on the backup server over time. The other alternative is to store each VM memory snapshot sequentially on disk, which results in slow random-access writes during normal operation but leads to smaller downtimes during failover because of faster sequential reads. Of course, this alternative is clearly preferable for solid state drives, since there is no difference between sequential and random write bandwidth. Considering these tradeoffs, in Yank's current implementation we use this design alternative to minimize downtime during failure.

3.5 Computing the Warning Time

Yank's correct operation depends on estimates of the advance warning time. There are multiple ways to compute the warning time. If transient servers connect to the grid, the warning time is static and based on server power consumption and UPS energy storage capacity. In the event of a power outage, if the UPS capacity is N watt-hours and the aggregate maximum server power is M watts, then the advance warning time is $\frac{N}{M}$. Alternatively, the warning time may vary in real-time if green data centers charge UPSs from on-site renewables. In this case, the UPSs' varying state-of-charge dictates the warning time, ensuring that transient servers are able to continue operation for some period if renewable generation immediately drops to zero. Note that warning time estimates do not need to be precisely accurate. Since, to minimize their amortized cost, data centers should not routinely use UPSs beyond a 40%-50% depth-of-discharge, even an "empty" UPS has some remaining charge to mind-the-gap and compensate for slight inaccuracies in warning time estimates. As a result, a natural buffer exists if warning time estimates are too short, e.g., by 1%-40%, although repeated inaccuracies degrade UPS lifetime.

4 Yank Implementation

Yank's implementation is available at http://yank.cs.umass.edu. We implement the snapshot manager by extending Remus inside the Xen hypervisor (v4.2). By default, Remus tracks dirty pages over short epochs (∼100ms) using shadow page tables and pausing VMs each epoch to copy dirty memory pages to a separate buffer for transmission to the backup server. While VMs may speculatively execute after copying dirty pages to the buffer, but before receiving

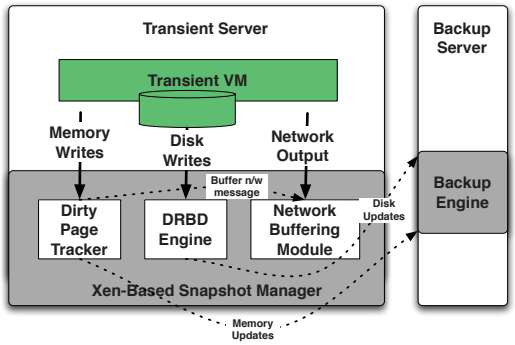


Figure 4: Snapshot Manager on the Transient Server

an acknowledgement from the backup server, they must buffer external network or disk output to preserve external synchrony. Remus only releases externally-visible output from these buffers after the backup server has acknowledged receiving dirty pages from the last epoch. Of course, by conforming to strict external synchrony, Remus enables a higher level of protection than Yank, including unexpected failures with no advance warning, e.g., fail-stop disk crashes. Although our current implementation only tracks dirty memory pages, it is straightforward to extend our approach to disk blocks.

Rather than commit dirty pages to the backup server every epoch, our snapshot manager uses a simple bitmap to track dirty pages and determine whether to commit these pages to the backup engine based on the upper and lower threshold, U_t and L_t. In addition, rather than commit CPU state each epoch, as in Remus, the snapshot manager only commits CPU state when it receives a warning that a transient server will terminate. Implementing the snapshot manager required adding or modifying roughly 600 lines of code (LOC), primarily in files related to VM migration, save/restore, and network buffering, e.g., xc_domain_save.c, xg_save_restore.h, and sch_plug.c. Finally, the snapshot manager includes a simple /proc interface to receive notifications about warnings or changes in the warning time. Figure 4 depicts the snapshot manager's implementation. As mentioned in the previous section, our current implementation uses DRBD to mirror disk state on a backup server.

Instead of modifying Xen, we implement Yank's backup engine "from scratch" at user-level for greater flexibility in controlling its in-memory queues and disk writing policy. The implementation is a combination of Python and C, with a Python front-end (∼300LOC) that accepts network connections and forks a backend C process (∼1500LOC) for each transient VM, as described below. Since the backup engine extends Xen's live migration and Remus functionality, the front-end listens on the same port (8002) that Xen uses for live migration. Figure 5 shows a detailed diagram of the backup engine.

For each transient VM, the backend C process accepts

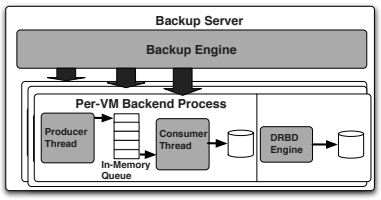


Figure 5: Backup Engine on the Backup Server

dirty page updates from the snapshot manager and sends acknowledgements. Each update includes the number of pages in the update, as well as each page's page number and contents (or a delta from the previous page sent). The process then places each update in an in-memory producer/consumer queue. To minimize disk writes, as described in Section 3.4.1, before queuing the update, the process checks the queue to see if a page already has a queued update. If so, the process merges the two updates. To perform these checks, the process maintains a hashtable that maps page numbers to their position in their queue. The process's consumer thread then removes pages from the queue (in LRU order based on a timestamp) and writes them to disk. In the current implementation, the backend process stores VM memory pages sequentially in a file on disk. For simplicity, the file's format is the same as Xen's format for storing saved VM memory images, e.g., via `xm save`. As discussed in Section 3.4.2, this results in low downtimes during migration, but lower performance during normal operation.

Finally, we implement Yank's restoration service (∼300LOC) at user-level in C. The daemon accepts a VM memory snapshot and an in-memory queue of pending updates, and then applies the updates to the snapshot without writing to disk. Since our implementation uses Xen's image format, the service uses `xm restore` from the resulting in-memory file to re-start the VM.

5 Experimental Evaluation

We evaluate Yank's network overhead, VM performance, downtime after a warning, and scalability, and then conduct case studies using real renewable energy sources. While our evaluation does not capture the full range of dynamics present in a production data center, it does demonstrate Yank's flexibility to handle a variety of dynamic and unexpected operating conditions.

5.1 Experimental Setup

We run our experiments on 20 blade servers with 2.13 GHz Xeon processors with 4GB of RAM connected to the same 1Gbps Ethernet switch. Each server running the snapshot manager uses our modified Xen (v4.2) hypervisor, while those running the backup engine and the restoration service use unmodified Xen (v4.2). In our experiments, each transient VM uses one CPU and 1GB RAM, and runs the same OS and kernel (Ubuntu 12.04, Linux kernel 3.2.0). We experiment with three benchmarks from Figure 3—TPC-W, SPECjbb, and a Linux kernel compile—to stress Yank in different ways.

TPC-W is a benchmark web application that emulates an online store akin to Amazon. We use a Java servlets-based multi-tiered configuration of TPC-W that uses Apache Tomcat (v7.0.27) as a front end and MySQL (v5.0.96) as a database backend. We use additional VMs to run clients that connect to the TPC-W shopping website. Our experiments use 100 clients connecting to TPC-W and performing the "browsing workload" where 95% of the clients only browse the website and the remaining 5% execute order transactions. TPC-W allows us to measure the influence of Yank on the response time perceived by the clients of an interactive web application.

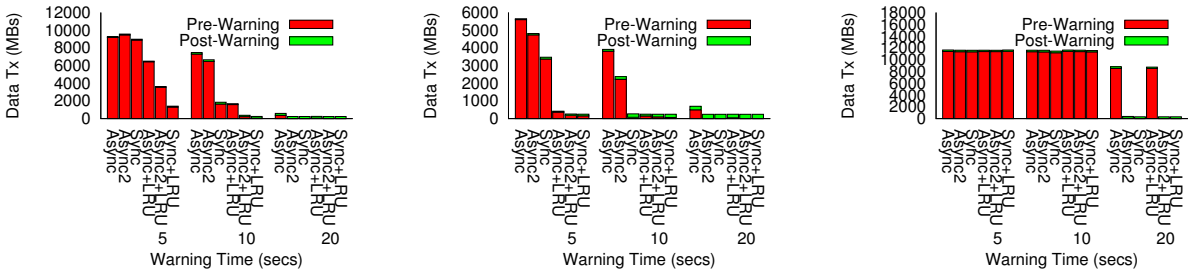
SPECjbb 2005 emulates a warehouse application for processing customer orders using a three-tier architecture comprising web, application, and database tiers. The benchmark predominantly exercises the middle tier that implements the business logic. We execute the benchmark on a single server in standalone mode using local application and database servers. SPECjbb is memory-intensive, dirtying memory at a fast rate, which stresses Yank's ability to maintain snapshots on the backup server without degrading VM performance.

Linux Kernel Compile compiles v3.5.3 of the kernel, along with all of its modules. The kernel compilation stresses both the memory and disk subsystems and is representative of a common development workload.

Note that the first two benchmarks are web applications, which are challenging due to their combination of interactivity and rapid writes to memory. We focus on interactive applications rather than non-interactive batch jobs, since the latter are more tolerant to delays and permit simple scheduling approaches to handling periodic power shortfalls, e.g., [2, 11, 12, 17]. Yank is applicable to batch jobs, although instead of scheduling them it shifts them to and from transient servers as power varies.

5.2 Benchmarking Yank's Performance

Network Overhead. Scaling Yank requires every transient VM to continuously send memory updates to the backup server. We first evaluate how much network traffic Yank generates, and how its optimizations help in reducing that traffic. As discussed in Section 3.3, the snapshot manager begins asynchronously committing dirty pages to the backup engine after reaching a lower threshold L_t. We compare this policy, which we call *async*, with the naïve policy, which we call *sync*, that enforces just-in-time synchrony by starting to commit dirty pages only after their size reaches the upper threshold U_t. Rather than commit all dirty pages when reaching U_t, which causes long pauses, the policy commits pages until the dirty pages reaches $0.9 * U_t$. We ex-



(a) Data Transferred with TPC-W (b) Data Transferred with Kernel Compile (c) Data Transferred with SPECjbb

Figure 6: Network overhead for each benchmark over a 15 minute period.

amine two variants of the async policy: the first one sets the lower threshold L_t to $0.5 * U_t$ and the second one sets it to $0.75 * U_t$. Our standard async and sync policies use a FIFO queue to select pages to commit to the backup engine; we label Yank's LRU optimization separately.

In this experiment, transient VMs draw power from the grid, and have a static warning time dictated by their UPS capacity. We also limit the backup engine to using an in-memory queue of 300MB to store memory updates from each 1GB transient VM. We run each experiment for 15 minutes before simulating a power outage by issuing a warning to the transient and backup server. We then measure the data transferred both before and after the warning for each benchmark. Figure 6 shows the results, which demonstrate that network usage, in terms of total data transferred, decreases with increasing warning time. As expected, the sync policy leads to less network usage than either async policy, since it only commits dirty pages when absolutely necessary. However, the experiment also shows that combining LRU with async reduces the network usage compared to async with a FIFO policy. We see that with just a 10 second warning time, Yank sends less than 100MB of data over 15 minutes for both TPC-W and the kernel compile, largely because after their initial memory burst these applications rewrite the same memory pages. For the memory-intensive SPECjbb benchmark, a 10 second warning time results in poor performance and excessive network usage. However, a 20 second warning time reduces network usage to $<$100MB over the 15 minute period.

Result: *The sync policy has the lowest network overhead, although async with LRU also results in low network overhead. With these policies, a 10 second warning leads to $<$ 100MB of data transfer over 15 minutes.*

Transient VM Performance. We evaluate Yank's effect on VM performance during normal operation by using the same experimental setup as before except that we do not issue any warning and only evaluate pre-warning performance. Here, we focus on TPC-W, since it is an interactive application that is sensitive to VM pauses from buffering network or disk output. We measure the average response time of TPC-W clients, while varying

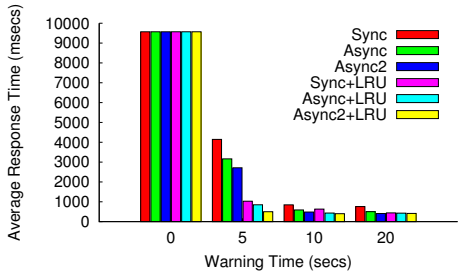


Figure 7: TPC-W response time as warning time varies.

both the warning time and the snapshot manager's policy for committing pages to the backup engine. Figure 7 shows that the async policy that selects pages to commit using and LRU policy results in the lowest average response time. The async policy reduces VM pauses, because the snapshot manager begins committing pages to the backup as soon as it hits the lower threshold L_t rather than waiting until reaching U_t, and forcing a large commit to the backup server. The experiment also demonstrates that with even a brief five second warning, the response time is $<$500ms using the async policy with LRU.

By contrast, with a warning time of zero the average response time rises to over nine seconds. In addition, the 90^{th} percentile response time was also near 15 seconds, indicating that the average response is not the result of a few overly bad response times. With no warning, the VM must pause and mirror every memory write to the backup server and receive an acknowledgement before proceeding. Although Remus [7] does not use our specific TPC-W benchmark, their results with the SPECweb benchmark are qualitatively similar, showing 5X worse latency scores. Thus, our results confirm that even modest advance warning times lead to vast improvements in response time for interactive applications.

Result: *Yank imposes minimal overhead on TPC-W during normal operation. With a brief five second warning time, the average response time of TPC-W is 10x less ($<$500ms) than with no warning time ($>$9s).*

VM Downtime after a Warning. The experiments above demonstrate that Yank imposes modest network and VM overhead during normal operation. In this experiment, we issue a warning to the transient server at

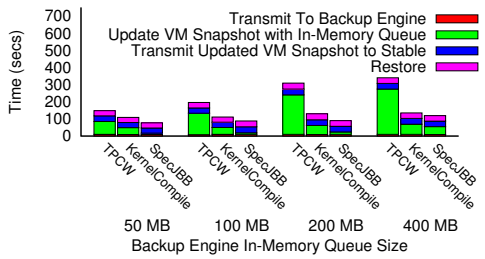


Figure 8: Downtime with the straightforward approach.

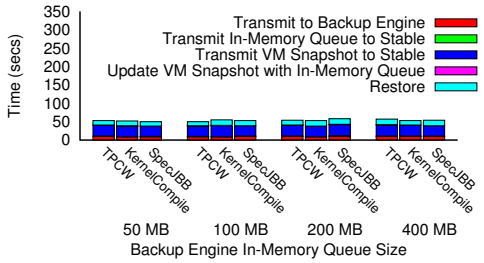


Figure 9: Downtime with Yank's optimizations.

the end of 15 minutes and measure downtime while the backup engine migrates the transient VM to a stable server. We compare Yank's approach (described in Section 3.4.1), which requires only a single read of the VM's memory image from disk, with a straightforward approach where the backup engine applies all updates to the memory snapshot before migrating it to the destination stable server. Note that in this latter case there is no need for Yank's restoration service, since the backup engine can simply perform a Xen stop-and-copy migration of the consistent memory snapshot at the backup server.

Figure 8 plots the transient VM's downtime using the straightforward approach, and Figure 9 plots it using Yank's approach. Each graph decomposes the downtime into each stage of the migration. In Figure 8, the largest component is the time required to create a consistent memory snapshot on the backup engine by updating the memory snapshot on disk. In addition, we run the experiment with different sizes of the in-memory queue to show that downtime increases with queue size, since a larger queue size requires writing more updates to disk after a warning. While reading the VM's memory snapshot from disk and transmitting it to the destination stable server still dominates downtime using Yank's approach (Figure 9), it is less than half than with the straightforward approach and is independent of the queue size. Note that Yank's downtimes are in the tens of seconds and bounded by the time to read a memory snapshot from disk. While these downtimes are not long enough to break TCP connections, they are much longer than the millisecond-level downtimes seen by live migration.

Result: *Yank minimizes transient VM downtime after a warning to a single read of its memory snapshot from disk, which results in a 50s downtime for a 1GB VM.*

Scalability. The experiments above focus on performance with a single VM. We also evaluate how many transient VMs the backup engine is able to support concurrently, and the resulting impact on transient VM performance during normal operation. Again, we focus on the TPC-W benchmark, since it is most sensitive to VM pauses. In this case, our experiments last for 30 minutes using a warning time of 10 seconds, and scale the number of transient VMs running TPC-W connected to the same backup server. We measure CPU and memory usage on the backup server, as well as the average response time of the TPC-W clients. Figure 10 shows the results, including the maximum of the average response time across all transient VMs observed by the TPC-W clients, the CPU utilization on the backup server, and the backup engine's memory usage as a percentage of total memory. The figure demonstrates that, in this case, the backup server is capable of supporting as many as 15 transient VMs without the average client response time exceeding 700ms. Note that without using Yank the average response time for TPC-W clients running our workload is 300ms. In addition, even when supporting 15 VMs, the backup engine does not completely use its entire CPU or memory.

Result: *Yank is able to highly multiplex each backup server. Our experiments indicate that with a warning time of 10 seconds, a backup server can support at least 15 transient VMs running TPC-W with little performance degradation for even a challenging interactive workload.*

5.3 Case Studies

The previous experiments benchmark different aspects of Yank's performance. In this section, we use case studies to show how Yank might perform in a real data center using renewable energy sources. We use traces from our own solar panel and wind turbine deployments, which we have used in prior work [4, 26, 27, 29]. Note that in these experiments the warning time changes as renewable generation fluctuates, since we assume renewables charge a small UPS that powers the servers.

5.3.1 Adapting to Renewable Generation

Figure 11 shows the renewable power generation from compressing a 3-day energy harvesting trace. For these experiments, we assume the UPS capacity dictates a maximum warning time of 20 seconds, and that each server requires a maximum power of 300W. Our results are conservative, since we assume servers always draw their maximum power. In the trace, at t=60, power generation falls below the 300W the server requires, causing the battery to discharge and the warning time to decrease. At t=80, power generation rises above 300W, causing the warning time to increase. Figure 11 shows the instantaneous response time of a TPC-W client running on a transient VM as power varies. The response time rises when

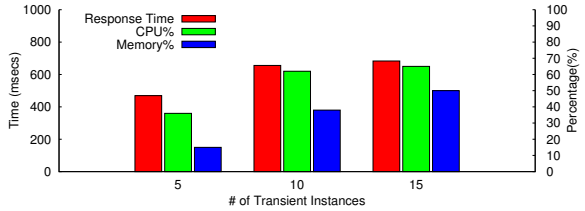


Figure 10: Yank scalability

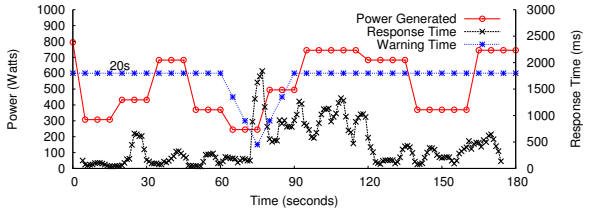


Figure 11: TPC-W response time as power varies

power generation decreases, and falls when it increases.

The experiment illustrates an important property of Yank: it is capable of translating variations in power availability to variations in application performance, *even for interactive application running on a single server*. Since servers remain far from energy-proportional, decreases in power availability require data centers to deactivate servers. Until now, the only way to scale application performance with power was to approximate energy proportionality in large clusters by deactivating a subset of servers [30]. For interactive applications not tolerant to delays, this approach is not ideal, especially if applications run on small set of servers. Of course, Yank's approach complements admission control policies that may simply reject clients to decrease load, rather than satisfying existing clients with a slightly longer response time. In many cases, simply rejecting new or existing clients may be undesirable.

Result: *Yank translates variations in power availability to variations in application performance for interactive applications running on a small number of servers.*

5.3.2 End-to-End Examples

We use end-to-end examples to demonstrate how Yank reacts to changes in renewable power by migrating transient VMs between transient and stable servers.

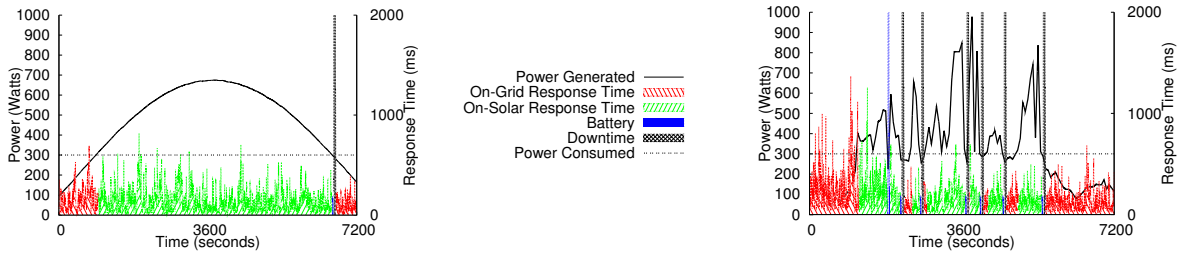
Solar Power. We first compress solar power traces from 7am to 5pm on both a sunny day and a cloudy day to a 2-hour period, and then scale the power level such that the trace's average power is equal to a server's maximum power (300W). While we compress our traces to enable experiments to finish within reasonable times, we do not introduce any artificial variability in the power generation, since renewable generation is already highly variable [31]. We then emulate a solar-powered transient server using a UPS that provides a maximum warning time of 30 seconds, although as above when power generation falls below 300W the UPS discharges and the warning time decreases. When the warning time reaches zero, Yank issues a warning and transfers the transient VM to a stable server. Likewise, when the warning time is non-zero continuously for a minute, Yank reactivates the transient server and transfers the VM back to it. Again, we run TPC-W in the VM and measure response time at the client as power generation varies.

Figure 12(a) and (b) shows that for both days Yank adapts to power variations with negligible impact on application performance. The sunny day (a) only requires the transient server to deactivate once, and has negligible impact on response time throughout the day. The cloudy day (b) requires the transient server to deactivate just seven times throughout the day. Thus, the application experiences seven brief periods of downtime, roughly 50 seconds in length, over the day. However, even though power is more intermittent than the sunny day, outside of these seven periods, the impact on response time is only slightly higher than during the sunny day. Note that, in this experiment, the periods where the VM executes on a stable server are brief, with Yank migrating the VM back to the transient server after a short time.

Wind Power. Wind power varies significantly more than solar power. Thus, in this experiment, we use a less conservative approach to computing the warning time. Rather than computing the warning time based on a UPS's remaining energy, we use a simple past-predicts-future (PPF) model (from [27]) to estimate future energy harvesting. The model predicts energy harvesting over the next 10 seconds will be same the same as the last 10 seconds. As above, we compress a wind energy trace from 7am to 5pm to two hours and scale its average power generation to the server's power. Since our PPF predictions operate over 10 second intervals, we use a UPS capacity that provides 10 seconds of power if predictions are wrong. We again measure TPC-W response time as it shifts between a transient and stable server.

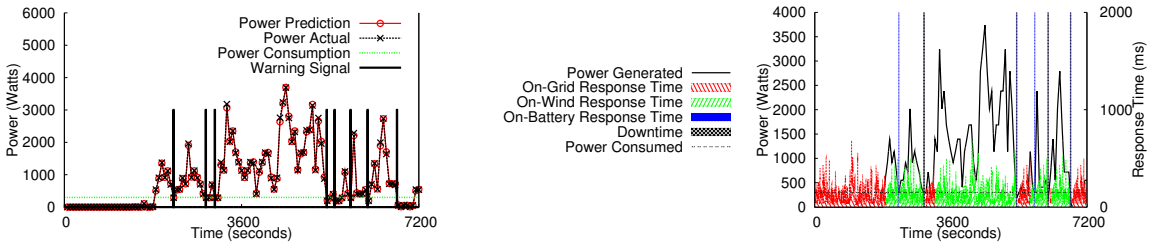
Figure 13(a) shows the PPF model accurately predicts power over these short timescales even though power generation varies significantly, allowing Yank to issue warnings with only a small amount of UPS power. Figure 13(b) shows the corresponding TPC-W response time, which is similar to the response time in the more stable solar traces. Of course, as during the cloudy day with solar power, when wind generation drops for a long period there is a brief downtime as the VM migrates from the transient server to a stable server.

Result: *Yank is flexible enough to handle different levels of intermittency in available power resulting from variations in renewable power generation.*



(a) Sunny Day (b) Cloudy Day

Figure 12: Yank using solar power on both a sunny and cloudy day.



(a) Short-term PPF prediction model (b) Wind-powered Server

Figure 13: Yank using wind power.

6 Related Work

Prior work on supporting renewables within a data center primarily targets non-interactive batch jobs, since these jobs are more tolerant to long delays when renewable power is not available. For example, GreenSlot [11] is a general batch job scheduler that uses predictions of future energy harvesting to align job execution with periods of high renewable generation. Similarly, related work [2, 12] proposes similar types of scheduling algorithms that specifically target MapReduce jobs. These solutions only support non-interactive batch jobs, and, in some cases, specifically target solar power [10], which is more predictable than wind power. Yank takes a different approach to support interactive applications running off renewable power. However, Yank's snapshots of memory and disk state are generic and also capable of supporting batch applications, although we leave a direct comparison of the two approaches to future work.

Recent work does combine interactive applications with renewables. For example, Krioukov et. al. design a power-proportional cluster targeting interactive applications that responds to power variations by simply deactivating servers and degrading request latency [17]. In prior work we propose a blinking abstraction for renewable-powered clusters, which we have applied to the distributed caches [26, 28] and distributed file systems [15] commonly used in data centers. However, blinking to support intermittent renewable energy requires significant application modifications, while Yank does not. iSwitch is perhaps the most closely-related work to Yank [18]. iSwitch assumes a similar design for integrating renewables into data centers, including some servers powered off renewables (specifically wind power) and some powered off the grid. However, iSwitch is more policy-oriented, tracking variations in renewable power to guide live VM migration between the two server pools. In contrast, Yank introduces a new mechanism, which iSwitch could use instead of live migration.

7 Conclusion

Yank introduces the abstraction of a transient server, which may terminate anytime after an advance warning. In this paper, we apply the abstraction to green data centers, where UPSs provides an advance warning, due to power shortfalls from renewables, move transient server state to stable servers. Yank fills the void between Remus, which requires no advance warning, and live VM migration, which requires a lengthy advance warning, to cheaply and efficiently support transient servers at large scale. In particular, our results show that a single backup server is capable of maintaining memory snapshots for up to 15 transient VMs with little performance degradation, which dramatically decreases the cost of providing high availability relative to existing solutions.

Acknowledgements. We would like to thank our shepherd, Ratul Mahajan, and the anonymous reviewers for their valuable comments, as well as Tim Wood and Navin Sharma for their feedback on early versions of this work. We also thank the Remus team, especially Shriram Rajagopalan, for assistance during early stages of this project. This research was supported by NSF grants CNS-0855128, CNS-0916972, OCI-1032765, CNS-1117221 and a gift from AT&T.

References

[1] S. Akoush, R. Sohan, A. Rice, A.W. Moore, and A. Hopper. Predicting the Performance of Virtual Machine Migration. In *MASCOTS*, August 2010.

[2] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing. Utilizing Green Energy Prediction to Schedule Mixed Batch and Service Jobs in Data Centers. In *HotPower*, October 2011.

[3] Apple. Apple and the Environment. http://www.apple.com/environment/renewable-energy/, September 2012.

[4] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, and J. Albrecht. Smart*: An Open Data Set and Tools for Enabling Research in Sustainable Homes. In *SustKDD*, August 2012.

[5] L. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[6] C. Clark, K. Fraser, S. Hand, J. G Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *NSDI*, May 2005.

[7] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. In *NSDI*, April 2008.

[8] P. Denning. The Working Set Model for Program Behavior. *CACM*, 26(1), January 1983.

[9] DRBD. DRBD: Software Development for High Availability Clusters. http://www.drbd.org/, September 2012.

[10] I. Goiri, W. Katsak, K. Le, T. Nguyen, and R. Bianchini. Parasol and GreenSwitch: Managing Datacenters Powered by Renewable Energy. In *ASPLOS*, March 2013.

[11] I. Goiri, K. Le, M. Haque, R. Beauchea, T. Nguyen, J. Guitart, J. Torres, and R. Bianchini. GreenSlot: Scheduling Energy Consumption in Green Datacenters. In *SC*, April 2011.

[12] I. Goiri, K. Le, T. Nguyen, J. Guitart, J. Torres, and R. Bianchini. GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks. In *EuroSys*, April 2012.

[13] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar. Benefits and Limitations of Tapping into Stored Energy for Datacenters. In *ISCA*, June 2011.

[14] S. Govindan, D. Wang, L. Chen, A. Sivasubramaniam, and B. Urgaonkar. Towards Realizing a Low Cost and Highly Available Datacenter Power Infrastructure. In *HotPower*, October 2011.

[15] D. Irwin, N. Sharma, and P. Shenoy. Towards Continuous Policy-driven Demand Response in Data Centers. *Computer Communications Review*, 41(4), October 2011.

[16] Jonathan Koomey. Growth in Data Center Electricity Use 2005 to 2010. In *Analytics Press*, Oakland, California, August 2011.

[17] A. Krioukov, S. Alspaugh, P. Mohan, S. Dawson-Haggerty, D. E. Culler, and R. H. Katz. Design and Evaluation of an Energy Agile Computing Cluster. In *Technical Report UCB/EECS-2012-13, EECS Department, University of California, Berkeley*, January 2012.

[18] C. Li, A. Qouneh, and T. Li. iSwitch: Coordinating and Optimizing Renewable Energy Powered Server Clusters. In *ISCA*, June 2012.

[19] H. Liu, C. Xu, H. Jin, J. Gong, and X. Liao. Performance and Energy Modeling for Live Migration of Virtual Machines. In *HPDC*, June 2011.

[20] Microsoft. Becoming Carbon Nneutral: How Microsoft is Striving to Become Leaner, Greener, and More Accountable. *Microsoft*, June 2012.

[21] R. Miller. Facebook Installs Solar Panels at New Data Center. In *Data Center Knowledge*, April 16 2011.

[22] U. F. Minhas, S. Rajagopalan, B. Cully, A. Aboulnaga, K. Salem, and A. Warfield. RemusDB: Transparent High Availability for Database Systems. In *VLDB*, August 2011.

[23] E. B. Nightingale, K. Veeraraghavan, P. M. Chen, and J. Flinn. Rethink the Sync. In *OSDI*, November 2006.

[24] S. Pelley, D. Meisner, P. Zandevakili, T. Wenisch, and J. Underwood. Power Routing: Dynamic Power Provisioning in the Data Center. In *ASPLOS*, March 2010.

[25] M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. *TOCS*, 10(1), February 1992.

[26] N. Sharma, S. Barker, D. Irwin, and P. Shenoy. Blink: Managing Server Clusters on Intermittent Power. In *ASPLOS*, March 2011.

[27] N. Sharma, J. Gummeson, D. Irwin, and P. Shenoy. Cloudy Computing: Leveraging Weather Forecasts in Energy Harvesting Sensor Systems. In *SECON*, June 2010.

[28] N. Sharma, D. Krishnappa, D. Irwin, M. Zink, and P. Shenoy. GreenCache: Augmenting Off-the-Grid Cellular Towers with Multimedia Caches. In *MMSys*, February 2013.

[29] N. Sharma, P. Sharma, D. Irwin, and P. Shenoy. Predicting Solar Generation from Weather Forecasts Using Machine Learning. In *SmartGridComm*, October 2011.

[30] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering Energy Proportionality with Non Energy-Proportional Systems – Optimizing the Ensemble. In *HotPower*, December 2008.

[31] Y.H. Wan. Long-term Wind Power Variability. Technical report, National Renewable Energy Laboratory, January 2012.

[32] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. Fathy. Energy Storage in Datacenters: What, Where, and How Much? In *SIGMETRICS*, June 2012.

[33] D. Williams, H. Jamjoom, Y. Liu, and H. Weatherspoon. Overdriver: Handling Memory Overload in an Oversubscribed Cloud. In *VEE*, 2011.

[34] T. Wood, 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 *SOCC*, October 2011.

[35] T. Wood, K. Ramakrishnan, P. Shenoy, and J. Van der Merwe. CloudNet: Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines. In *VEE*, March 2011.