Activity Based Sector Synchronisation: Efficient Transfer of Disk-State For WAN Live Migration

Sherif Akoush, Ripduman Sohan, Bogdan Roman, Andrew Rice and Andy Hopper
University of Cambridge Computer Laboratory
firstname.lastname@cl.cam.ac.uk

***Abstract*—Live migration of virtual machines is now commonplace but the issue of synchronising storage remains an obstacle to wide-area migration between datacentres. We discuss the range of possible designs for synchronising disk images and argue that we have to be adaptive at the sector level to efficiently transfer disk-state to the destination. We show, using a number of production workloads, that delaying the transmission of "hot" sectors can achieve considerable bandwidth reduction over a naïve eager strategy. We introduce the Activity Based Sector Synchronisation (ABSS) algorithm and show by trace-driven simulations that ABSS is able to deliver a useful trade-off with a data transmission overhead between 0.3% and 2.4% of a gigabit link whilst incurring in the majority of cases a migration latency of less than 1.1 s.**

I. Introduction

Virtualisation technology brings key advantages to datacentre operators [1] such as service consolidation, increased hardware utilisation, stronger application isolation and simplified resource allocation. This efficient management of the computing infrastructure result in significant cost reductions for corporate environments.

Live migration [2] is a fundamental feature of modern virtualisation platforms which enables the seamless movement of running virtual machines (VMs) between hosts, decoupling them from physical hardware. It is commonly used to relocate VMs in one datacentre to load balance resources, compact clusters in periods of low demand and perform server maintenance without violating service level agreements (SLAs).

The current generation of live migration architectures are designed around the premise that migrations only occur between hosts colocated in the same datacentre. Consequently, these architectures assume that the underlying disk images are universally accessible by all hosts using a low-latency high-throughput network. Consequently, migrations are reduced to the transfer of CPU and memory-state.

However, four orthogonal trends in enterprise virtualisation environments indicate that inter-datacentre migrations will be desirable in the near future. (*i*) Scaling: As the size of virtualisation installations grow to multiple physical locations, load balancing and cluster consolidation may require migrations across datacentres. (*ii*) Increasing resilience: Organisations guarantee reliability of critical services by mirroring important VMs across geographically distant nodes. (*iii*) Load balancing: Cloud computing allows organisations to dynamically allocate resources according to demand. (*iv*) Standardisation[1]: Cloud platforms should soon make it standard to migrate VMs between different cloud services. These trends may not be complementary but they all require the application of wide-area network (WAN) scale migrations of VMs.

Inter-datacentre migration would naturally require that disk images are *locally* accessible for VMs to continue running without performance degradation. Therefore, we need a mechanism to *synchronise* disk images at locations where the VM could be relocated. However, efficiently synchronising disk images is challenging especially over the high-latency low-bandwidth WAN links that typically interconnect datacentres.

The large size of VM disk images means that it is impractical to synchronise the replica at the point of migration due to the big required to transfer all modified sectors. Conversely, copying sectors as soon as they are modified by the VM is ineffectual and inefficient if they are shortly re-modified (as bandwidth is wasted in this case).

This paper introduces *Activity Based Sector Synchronisation* (ABSS), a technique for the *efficient* and *timely* synchronisation of running VM disk images over WAN. The design of the algorithm is based on observations from recent production server workloads. ABSS achieves efficiency by transferring written sectors only when they are unlikely to be modified in the near future thereby conserving bandwidth. On the other hand, ABSS quickly identifies these sectors thereby minimising its latency overhead at the point of migration. ABSS can reduce the amount of data transferred considerably while incurring in the majority of cases a small migration overhead less than 1.1 s on gigabit links.

The rest of the paper is organised as follows. Section II outlines the challenge of achieving disk synchronisation in detail. Section III characterises how workloads write to disk by analysing real-world production server traces. Based on this study, we define the parameters that are important to design an efficient storage synchronisation algorithm. Section IV presents ABSS and how it can be used to transfer disk-state to the remote host using per-sector statistics. Section V evaluates ABSS against other algorithms with respect to the amount of data transferred and the migration latency incurred. Section VI shows the implementation issues of ABSS. Finally we discuss related work, future directions and then conclude.

[1] http://opencloudconsortium.org/

TABLE I
AMAZON EC2 STANDARD INSTANCES

Instance Type	Property	
	Memory (GB)	Disk (GB)
Small	1.7	160
Large	7.5	850
Extra Large	15	1,690

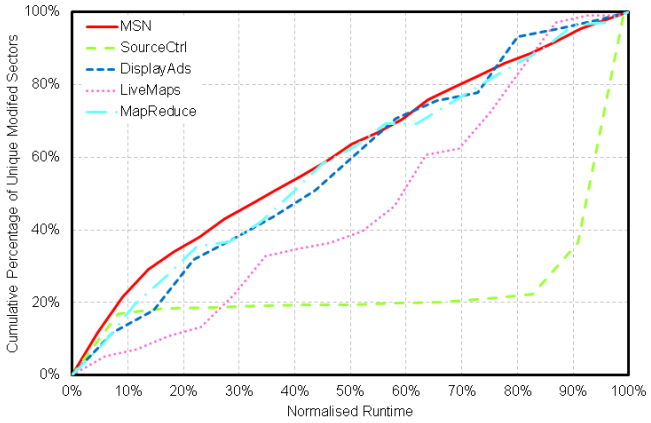

Fig. 1. Cumulative Percentage of Unique Sectors Modified with Respect to Time.

II. THE CHALLENGE OF DISK-STATE SYNCHRONISATION

Current live migration architectures assume that VM disk images are stored on shared network storage devices and are universally accessible by all hosts. This is a reasonable assumption as these hosts are located in the same datacentre.

However, universal storage accessibility does not scale across datacentres separated by large geographic distances and connected over high-latency low-bandwidth links. In this scenario, accessing remote disk images is slow and resource intensive due to the requirement that every operation be transmitted across the network.

For VMs to continue functioning with acceptable performance after an inter-datacentre migration, it is necessary that a synchronised copy of the VM disk image is locally accessible at the destination. However, the large size of VM disk images means that synchronisation may be a protracted process. Table I lists the memory and disk sizes for Amazon EC2[2] standard instance types at the time of writing. Assuming a simple single-pass synchronisation that iterates through the entire address space, it would require 1–2 orders of magnitude longer to transfer disk-state compared to memory.

Therefore, the design of the disk image synchronisation algorithm significantly influences the amount of data required to be transferred and the VM migration time. The simplest possible algorithm is one that, at the point of migration, transfers sectors that are either missing or outdated at the destination host (on-migration synchronisation). While this design is optimal with respect to the amount of data transferred (as only the last update of a given sector is sent), the time required to execute a migration is proportional to the number of sectors that have been modified and hence require to be transferred to the destination. As explained previously, the nature of datacentre interconnects can render this approach too slow for production deployment. Our analysis of five workloads (details of which are available in Section III) indicates that on-migration synchronisation would suffer a significant latency overhead. This is exemplified in Figure 1 showing that there is an increasing set of modified sectors directly proportional to VM runtime for all workloads.

In order to mitigate the large latency associated with on-migration synchronisation, we advocate the need to synchronise disk images before migration is scheduled. In other words, we propose to transfer disk-state changes in a way that is decoupled from the migration process with the expectation that when the actual migration is needed, we will not have a huge backlog of unsynchronised sectors. Consequently, we reduce the migration latency overhead.

In this scenario, synchronisation can be achieved by transferring every sector as soon as it is modified (on-modification synchronisation). The source and destination images are always up-to-date eliminating the potentially large migration time associated with on-migration synchronisation. However, on-modification synchronisation's data transfer requirement is directly proportional to the number of write operations. This relationship is disadvantageous in circumstances where the financial cost of the link is correlated with its usage, as in the case with standard interconnects' pricing models. Moreover, the energy consumption of the network is directly related to the number of packets sent [3]. Coupled with the current emphasis on reducing the energy footprint and carbon emissions of networks, on-modification synchronisation is not ideal.

In this paper, we propose an algorithm that provides a good trade-off between the bandwidth required to synchronise disk-state and the latency incurred when migration is triggered. We base the design of our algorithm on the characteristics of how workloads modify disk sectors. The next section analyses five applications to obtain this information.

III. CHARACTERISING WRITE ACTIVITY

In this section, we analyse block-level traces of five workloads to understand how typically applications modify disk.

Table II and Table III provide a synopsis of the studied workloads and the mnemonic identifiers used in this paper. We select five workloads to do our analysis while we choose another five for the evaluation of our algorithm. Hence, we hope that our results apply to a wider set of applications.

Four workloads (LiveMaps, DisplayAds, SourceCtrl and MSN) were selected from public logs recently made available by Microsoft on the SNIA trace repository.[3] Information about these workloads is available in the papers [4], [5] that describe them. These papers only provide high level read and write activity patterns but we extract more detail about the write behaviour for the purpose of our study. A fifth workload

[2]http://aws.amazon.com/ec2/

[3]http://iotta.snia.org/

TABLE II
SET OF WORKLOADS FOR ANALYSIS

Workload Mnemomic	Description	Year	Duration
MSN	MSN storage metadata	2008	6 hours
LiveMaps	Map imagery repository	2008	24 hours
DisplayAds	Caching tier for Ads	2008	24 hours
SourceCtrl	Source control	2007	7 days
MapReduce	Map-reduce tasks	2010	4 hours

TABLE III
SET OF WORKLOADS FOR EVALUATION

Workload Mnemomic	Description	Year	Duration
MSNBE	MSN user files repository	2008	6 hours
Payload	Ads selection	2008	24 hours
Proxy	Firewall/web proxy	2007	7 days
RadiusAuth	Corporate remote access	2007	18 hours
RadiusBE	Radius SQL back-end	2007	18 hours

(MapReduce) was generated in-house and augments the Microsoft set by providing traces of write activity generated by MapReduce [6] tasks (running on Hadoop[4]) that are processing basic HTML documents [7]. The experimental setup for this workload is available in prior work [8].

These workloads were picked for their diversity as they represent a wide range of write activity patterns. For example, LiveMaps has a high write rate, one order of magnitude more than DisplayAds. Additionally, MSN presents a more homogenous disk activity than the others. SourceCtrl is a seven week log that highlights features that are not seen in the other relatively shorter traces. Finally, MapReduce is considered a useful workload as there is currently widespread interest in using this computation paradigm for large-scale data processing and analysis [9], [10] in various fields such as web search, data mining and machine learning. Moreover, the MapReduce workload provide a different operating and file system than the ones used in the Microsoft workloads.

We studied the five workloads in order to determine the temporal characteristics of write operations. Noteworthy results are summarised below.

A. Write rate is low

Table IV details the write rate (the amount of data written per second) observed in the traces. The average write rate across all workloads is 1.61 MB/s, which is relatively low compared to gigabit links. The small standard deviation (less than 2 MB/s) for all (except LiveMaps) further provides support that write rates are consistently low. Similarly, the 99%-ile for four out of five workloads is less than 7MB/s. Finally, the maximum value is less than 30 MB/s except for LiveMaps. However, LiveMaps exceeds this rate only for a combined two seconds span. Assuming gigabit interconnects, this result suggests that synchronisation across datacentres is feasible.

B. Sector writes are unbalanced

Figure 2 shows the cumulative percentage of the number of updates to the same sector for all five workloads. At least

[4]http://hadoop.apache.org/

TABLE IV
WRITE RATE FOR ALL WORKLOADS

Metric (MB/sec)	Workload				
	LiveMaps	MSN	DisplayAds	SourceCtrl	MapReduce
Avg.	6.61	0.66	0.01	0.05	0.70
Std. Dev.	8.71	0.99	0.23	0.33	1.65
99%-ile	36.01	4.21	0.12	0.28	6.95
Max.	169.55	22.48	17.96	23.36	28.34

TABLE V
WRITE OPERATIONS FOR ALL WORKLOADS

Workload	Operation Type	
	Initial Write	Overwrite
MSN	31.64%	68.36%
LiveMaps	99.39%	0.61%
DisplayAds	12.34%	87.66%
SourceCtrl	15.92%	84.08%
MapReduce	48.64%	51.36%

90% of sectors are written four times or less (in LiveMaps specifically, 99% of sectors are modified just once). Hence, we assume that most sectors are changed only a few times.

However, the remaining 10% of sectors suffer more than 50% (with the exception of LiveMaps) of the total write operations as illustrated in Table V. It follows logically that overwrites happen on a small set of sectors. This long tail behaviour may notably increase the amount of data transferred in on-modification synchronisation algorithm due to the requirement that remodified sectors have to be resent.

C. Overwrite operations are clustered

We have noticed that sectors are generally modified in *clusters* (groups) of writes. In other words, a given sector repeatedly enters a period of high activity (when it is being modified frequently) followed by a period of low activity. This observation is intuitively related to the way applications typically behave. Consider the case of a web server that has daily, weekly and even seasonally varying patterns; this periodic load directly influences all sub-systems including storage. At periods of high request rates (that modify persistent-state information), the corresponding files backed up by some sectors are modified repeatedly. When the load goes down (e.g. during night time), these sectors are no longer being updated.

To substantiate this behaviour, we ran a gradient-based cluster detection algorithm on all sectors that are modified at least twice for all five workloads. Figure 3 illustrate the weighted (more weight to sectors that are frequently modified) percentage of how many write operations fall within a given time boundary. Obviously as we increase the time boundary, we observe more operations grouped together. Results show that more than 90% of writes happen within ten minutes intervals for the majority of workloads. Consequently, we assume that writes to a given sector happen generally in clusters (groups). Additionally, we observe that write clusters are spaced on a sector; i.e. for a window of 4,000 s, more than 95% of the writes are grouped together.

Finally, we argue that the behaviour of the write activity for a given sector may change with time. Applications have varying patterns, which directly influence the usage of the storage sub-system. Generally, we expect that overwrites are

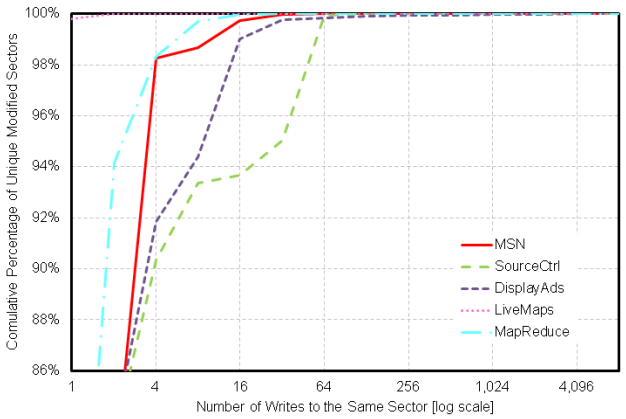

Fig. 2. Cumulative Percentage of the Number of Writes to a Specific Sector

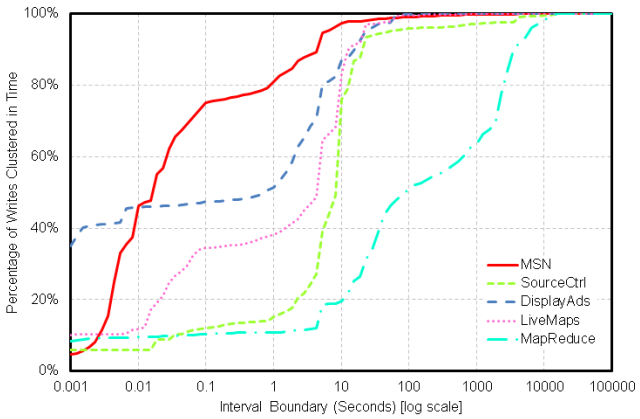

Fig. 3. Writes that Are Clustered in Time for Different Boundaries

clustered, spaced and their distribution might change over time.

D. Sectors are written in 4KB blocks

We analysed the temporal and spatial locality across sectors by counting the number and address of additional sectors modified on *every* write event over time intervals ranging from 10 ms to 1 s. Our results indicate that write operations are at least 8 sectors long (an expected result as most filesystems default to a 4KB logical block size for the purpose of efficiently aligning memory pages and disk blocks).

E. Conclusion

The results of this section advocate the need to design a dynamic synchronisation algorithm that works at the sector level and takes write patterns into account. We describe it in the next section.

IV. Activity Based Sector Synchronisation

A. Drawbacks of Trivial Synchronisation Algorithms

Our analysis in the previous section showed that a simple algorithm based on the transfer of sectors as soon as they are modified, will lead to unnecessary data transfer and inefficient use of the link. Table V illustrates that overwrite operations account for more than 50% in almost all workload. From this perspective, on-modification is not optimal.

Another trivial approach is periodic synchronisation that only transfers modified sectors at repeated intervals (e.g. every ten minutes). This design reduces the amount of data transferred as it only sends the most recent copy of modified sectors since the last synchronisation event. Hence, overwrites to the same sector are "absorbed" during one period.

The effectiveness of this design is dependent on the static period between synchronisation events. Using a small period keeps the two disk images almost synchronised; but bandwidth is wasted in cases where overwrites are frequent. On the other hand, having a large period needlessly increases the amount of data that needs to be transferred at migration and leads to undesirable bursty link usage. Moreover because periodic synchronisation treats all sectors equally, (*i*) it will wastefully retransmit sectors that are frequently updated and (*ii*) it will unnecessarily delay sectors that are written once. Therefore it is more efficient to be adaptive at the sector level.

B. Overview of ABSS

We now outline *Activity Based Sector Synchronisation* (ABSS), a statistically based synchronisation algorithm designed to minimise both the amount of data transferred and the VM migration time. The principles of ABSS are based on observations drawn from the workload analysis presented in Section III. While it may appear that an algorithm based on the study of just five workloads will have limited applicability, we believe that the diversity in our chosen set renders the algorithm suitable to a wider range of production applications.

ABSS is based on three principles: (*i*) maintaining statistics and performing synchronisation at the sector level, (*ii*) using previously observed sector modification times to predict when it is likely to be updated, and (*iii*) only synchronising sectors that are unlikely to be modified in the near future.

In effect, the algorithm uses previously observed write intervals (the time difference between two consecutive writes to the same sector) to create an adaptive statistical model that determines when a localised cluster of writes ends and only then it synchronises the sector. In doing so, ABSS minimises the data transfer requirement by avoiding synchronisation during a period of high write activity. Similarly, the algorithm reduces the VM migration latency by transferring the sector as soon as it enters a period of inactivity. Figure 4 illustrates a typical behaviour of ABSS on a given sector. The remainder of this section outlines the technical details of the algorithm.

C. Identifying Write Cluster Boundaries

Write cluster boundary identification is a fundamental function of ABSS. Accurate prediction of these boundaries (i.e. periods of inactivity in which the sector is unlikely to be changed) enables timely and efficient synchronisation of modified sectors. The ABSS algorithm is based on the assumption that overwrite events happen in clusters (groups) as discussed previously in Section III-C.

The basic operation of the ABSS cluster boundary identification algorithm is simple: provided a set of previously observed modification timestamp events for a given sector,

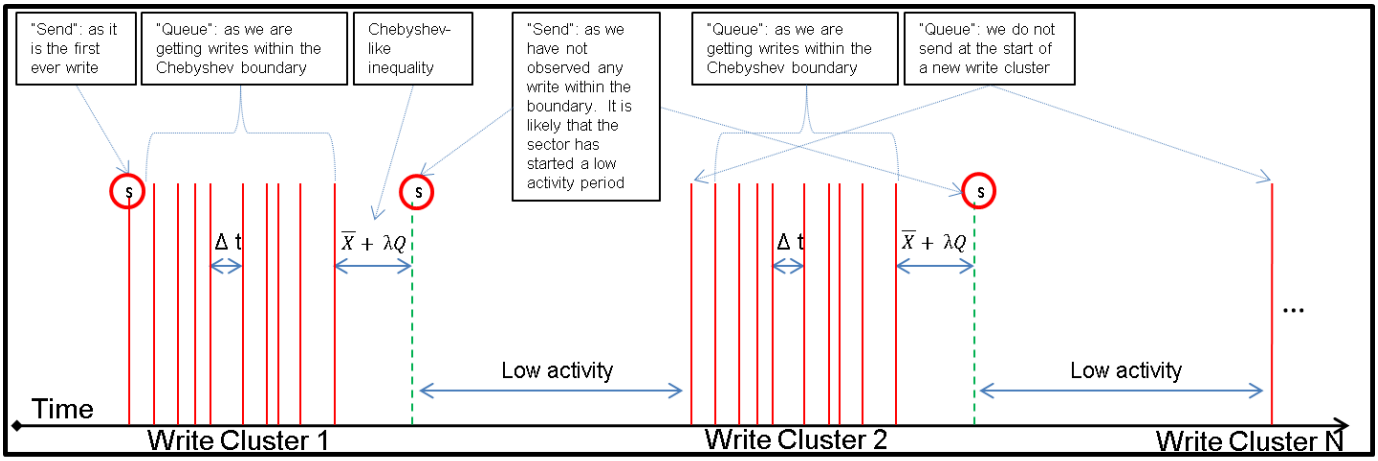

Fig. 4. Schematic Diagram of ABSS Behaviour on a Given Sector

ABSS calculates the theoretical *temporal upper bound* limit for *any* future write events that are predicted to happen during the current activity period. Should an event *not* be observed before the upper time limit is reached, it is highly likely that this sector has entered a low activity period and hence it is synchronised. Alternatively should a write be observed, it is incorporated into a recalculation of the upper bound.

We use a Chebyshev-type inequality (introduced by Saw et al. [11]) to identify write cluster boundaries based on maintaining the sample mean and variance of the write intervals. Formally, given $n \geq 2$ weakly exchangeable[5] samples $X_1, X_2, X_3, \cdots, X_n$ from some unknown distribution, the sample mean $\overline{X} = \sum_1^n X_j/n$, the sample variance $S^2 = \sum_1^n (X_j - \overline{X})^2/(n-1)$, $Q^2 = (n+1)S^2/n$, $\lambda \geq 1$ and v the largest integer less than $(n+1)(n-1+\lambda^2)/(n\lambda^2)$ then:

$$P\{|X - \overline{X}| \leq \lambda Q\} > 1 - v/(n+1) \quad (1)$$

which provides a prediction interval for the next observation. In our case, by maintaining the sample mean ($\overline{X}$) and sample variance (S^2) of the time difference between observed consecutive writes we can estimate the temporal upper bound (as measured from the timestamp of the last write operation) for events occurring in this distribution using (1), where the right hand side expresses the required confidence interval (c).

To achieve a specific c (say 90%), we have to tune λ in (1) for each sample size n. Intuitively for small n, λ is relatively large in order to capture the uncertainty regarding the estimates of the current sample. As we see more data points, λ starts to decrease; and as n goes to infinity, the limit tends towards the standard Chebyshev bound $1 - 1/\lambda^2$ [11].

A cluster boundary is considered identified if there is no subsequent write operation until the upper time bound from the last operation is reached. As soon as a boundary has passed, the sector is synchronised. Varying the confidence interval parameter c enables a trade-off between bandwidth and latency; smaller values reduce migration latency at the cost of potentially higher data transfer requirement while larger values increase migration latency but can minimise data transfer requirement.

By design ABSS incorporates every write interval of the current activity into the model as the mean and variance over these observed events are recalculated. Having said that, a single outlier event has little impact but a change in the distribution is eventually captured.

D. Synchronising Single-Write Sectors

As the analysis in Section III-B showed, the vast majority of sectors are modified just once. Building a write cluster model for these sectors is a futile task as it only serves to increase the migration latency. To account for this behaviour, ABSS reverts to on-modification synchronisation for all sectors for which it does not possess prior information; i.e. the first time a sector is written, it is synchronised immediately. While this design choice will result in some wasted data transfer for sectors that are subsequently written, we believe it offers a good trade-off between the data transfer requirement and the migration latency overhead.

E. Synchronising Periodically Updated Sectors

Periodically updated sectors that are modified at regular intervals also require special treatment. As their cluster boundary are continually extended on every write operation, the upper boundary will never be reached and so these sectors will never be synchronised. While sectors with such write distributions are likely to be few and far between, nevertheless ABSS is required to handle them.

The ABSS logic for synchronising periodically updated sectors is based on the observation that the likelihood a synchronised sector is up to date, is proportional to the sector write intervals. Thus, transferring sectors with large write intervals is likely to reduce the VM migration time without significantly increasing the synchronisation data transfer requirement.

ABSS therefore synchronises all sectors that have remained unmodified for a set period of time, regardless of whether their upper time bound has been reached. We have noticed in Section III-C that more that 95% of the writes are grouped

[5]The weak exchangeability condition holds in our case [11].

Algorithm 1 Sector Statistics Modelling

```
1:  procedure UPDSECTSTATS(secNum,trackedSectors)
2:      curTime ← writeEvent.getTime()              ▷ Time of the write event
3:      if secNum ∉ trackedSectors then
4:          s ← Sector()                            ▷ Create new sector data structure
5:          s.secNum ← secNum
6:          s.curWrCnt ← 0
7:          s.curMean ← 0.0
8:          s.curStDev ← 0.0
9:          s._curVar ← 0.0                         ▷ Private variable for variance
10:         s.lastWrTime ← 0
11:         s.newlyTrackedSector ← True             ▷ Flag new sector
12:         trackedSectors ← trackedSectors ∪ {s}
13:     end if
14:     s ← {s | s ∈ trackedSectors ∧ s.secNum = secNum}
15:     s.curWrCnt ← s.curWrCnt + 1
16:     if s.curWrCnt > 1 then
17:         wrDelta ← curTime − s.lastWrTime
18:         mDelta ← wrDelta − curMean
19:         curMean ← s.curMean
20:         s.curMean ← curMean + mDelta / (s.curWrCnt − 1)
21:         s._curVar ← s._curVar + mDelta × (wrDelta − s.curMean)
22:         s.curStDev ← SQRT(s._curVar / (s.curWrCnt − 1))
23:     end if
24:     s.lastWrTime ← curTime
25: end procedure
```

Algorithm 2 Sector State Determination

```
1:  function DETSECTSTATE(secNum,trackedSectors,
                          unmodTimeThreshold,c)
2:      s ← {s | s ∈ trackedSectors ∧ s.secNum = secNum}
3:      curTime ← getCurrentTime()
4:      if s.newlyTrackedSector then
            return EVENT_SECTOR_FIRST_WRITE
5:      end if
6:      if curTime − s.lastWrTime > unmodTimeThreshold then
            RESETSECTORSTATS(s)
            return EVENT_SECTOR_UNMOD_MAXTTHRESH
7:      end if
8:      if s.curWrCnt > 1 then
9:          λ ← lambdas[s.curWrCnt − 1,c] ▷ Get λ according to the number of
        data points in the sample and the confidence interval
10:         Q ← ((s.curWrCnt)/s.curWrCnt − 1)^{1/2}× s.curStDev
11:         threshold ← λ × Q
12:         if curTime − s.lastWrTime> s.curMean + threshold then
              return EVENT_UPPER_TIME_BOUNDARY_EXCEEDED
13:         end if
14:     end if
            return NO_EVENT
15: end function
```

together for a time window of 4,000 s (≈ one hour). Consequently, setting the unmodified threshold to this arbitrary value is likely to optimise the data transfer requirement. However, we acknowledge that it may be advantageous to consider this threshold a workload-specific tunable parameter.

While we have illustrated the problem of an always extending cluster boundary using the specific example of periodically updated sectors, it is important to note that this issue can manifest itself on any sector where its modification pattern results in subsequent writes occurring before the upper time bound is reached. In this case, the unmodified threshold performs the same function as well.

F. The ABSS Algorithms

ABSS can be implemented as a set of complementary algorithms, one for modelling sector statistics and the other for determining sector state. Algorithm 1 provides a pseudocode implementation for the modelling algorithm, written to promote clarity over efficiency. As illustrated, the algorithm receives two input parameters: `secNum` and `trackedSectors`, which identify the sector number and the set of tracked sectors respectively. If the provided sector is not being tracked it is initialised, added to the set of tracked sectors (lines 4–12) and its modification details are updated. The mean and variance calculations (lines 16–22) are done on the fly using the Welford method [12].

The sector state determination algorithm (Algorithm 2) receives two additional parameters: `unmodTimeThreshold` determines the time threshold after which a sector is synchronised regardless of the upper time bound (Section IV-E) and `c` provides the required confidence interval (Section IV-C).

Algorithm 2 signals three important states to synchronise sectors: (*i*) The first time the sector is written in agreement with the single-write case (lines 4–5); (*ii*) a sector is unmodified for a set amount of time regardless of its upper bound (lines 6–7); (*iii*) finally, if no overwrite event has occurred until the upper time bound is reached (lines 12–13). These design choices have been illustrated previously.

Additionally, the Chebyshev-type bound is computed according to the required confidence interval, the sample mean, the sample variance and the number of overwrites in the current activity (lines 9–11). Effectively we pre-compute the different values of λ using the right hand side of (1) and store them in a lookup table that is accessed in $O(1)$ at runtime.

V. EVALUATION

This section evaluates ABSS to characterise its effectiveness in optimising both the amount of data transferred and VM migration time. We compare ABSS against the best-of-breed algorithms (on-migration, on-modification and periodic synchronisation). The evaluation is carried out using simulations on a different set of server traces than those used in the analysis, to counter any confirmation bias that may have arisen regarding the design choices of ABSS. These workloads were also selected from public logs made available by Microsoft on the SNIA trace repository. However, they comprise a completely different set of applications as outlined in Table III. In all simulations, we assume the link speed is 1Gbps.

The evaluation focuses on three main points: (*i*) Comparing ABSS's data transfer and VM migration time metrics to on-modification and periodic synchronisation, (*ii*) outlining the effect of ABSS's tunable parameters on its performance, and (*iii*) analysing the behaviour of ABSS on write-once and overwrite dominant workloads (extreme cases).

Our preliminary evaluation showed that on-migration synchronisation provided the worst results for the migration latency overhead as, in some cases, more than 250GB of disk

TABLE VI
PERFORMANCE OF PERIODIC SYNCHRONISATION (PERIOD = 4,000 S) RELATIVE TO ABSS (C=90% AND UNMODTIMETHRESHOLD=4,000 S)

Workload	Bandwidth	Latency
MSNBE	-8.8%	94.8%
Proxy	19.4%	83.6%
Payload	-3.8%	761.1%
RaduisAuth	-8.4%	899.9%
RaduisBE	-5.9%	824.8%

TABLE VII
MIGRATION LATENCY OVERHEAD FOR ABSS ASSUMING 1GBPS LINK

Workload	Avg. (4KB Sector)	Avg. (ms)	Std. Dev. (ms)
MSNBE	844,519.3	25,772.7	10,713.4
Proxy	15,153.3	462.4	50.0
Payload	35,434.9	1,081.4	323.3
RaduisAuth	4,920.0	150.1	116.0
RaduisBE	18,610.7	567.9	352.9

sectors are required to be transferred at the point of migration. For this purpose, we eliminated it from further analysis.

A. Comparison with Other Algorithms

1) The Data Transfer: We examine the amount of data that is required to be transferred by ABSS for the purpose of maintaining disk synchronisation and compare it with on-modification and periodic synchronisation. In order to be considered viable, at an equivalent point of migration, ABSS should have transferred less data than either of the other two algorithms. To do so, we compare the average amount of data sent per second for on-modification, periodic (utilising periodic values of 32, 64 and 128 s), and ABSS (using 90% and 4,000 s for the required confidence interval and the unmodified time threshold respectively).

Figure 5 provides the results for this comparison normalised to the worst performing algorithm (on-modification synchronisation). ABSS has an average lower data transfer overhead in all cases, transmitting between 9% and 35% less data than the second best algorithm. In Proxy particularly, ABSS transmits 35% less data than the next best algorithm and 96% less than on-modification synchronisation. The excellent results seen in Proxy are due to the property that the workload has a very predictable update pattern. A web proxy typically refreshes local caches for the frequently accessed sites with a set periodicity, which is directly affecting the way sectors are being modified. ABSS can easily capture this behaviour, which effectively bounds when a sector is likely to get overwritten and decides correctly whether to keep or sent it. However, for workloads composed of mainly single writes (e.g. RadiusAuth), the results are less pronounced. Nevertheless, on average ABSS transmits 20% less data than the next best algorithm and 37% less than on-modification synchronisation.

The absolute values of the average data transferred per second of ABSS are overlayed on Figure 5, which show that our algorithm has a low badwidth requirement for all workloads. The highest value is 3167.6 KB/s observed for MSNBE while we can achieve as low as 41.6 KB/s for Proxy. Assuming 1Gbps interconnects, ABSS uses on average between 0.3% and 2.4% of the link capacity to synchronise sectors to the destination. Hence, our algorithm enables the synchronisation of hundreds or even thousands of VMs simultaneously.

Data transfer variance is considerably lower in ABSS. In the cases of MSNBE and Payload for example, our algorithm reduces the throughput variance by more than 31% and 65% respectively compared to the next best algorithm (on-modification synchronisation). One notable exception is Proxy; ABSS has minimised its average throughput aggressively in the expense of higher variance. Generally ABSS is designed to absorb overwrites, it is expected generally to have a smoother throughput than the other algorithms.

To further quantify the savings of ABSS, we evaluate it against a theoretical "best-case" algorithm that only synchronises the last update of write clusters of a given sector. This algorithm is designed to absorb as much overwrite operations as possible thus minimising data transfer. Time between clusters is chosen to be 4,000 s. Figure 5 shows that ABSS is comparable to the best-case (between 70% and 99%). We believe that any further improvements on our algorithm should be workload-specific and involve distribution-related optimisations. We note as well that the best-case algorithm might (and in practice will) incur a higher VM migration time.

Additionally, we compare periodic synchronisation having a value of 4,000 s with ABSS. Table VI shows the bandwidth and migration latency increase for this large period. While the bandwidth requirements for both algorithms are similar, the migration latency for periodic synchronisation is 1–2 order of magnitude more than ABSS. This further proves that being adaptive at the sector level is advantageous.

2) The VM Migration Time: In order to characterise VM migration time, we measure the average number of 4KB sectors *held* (not synchronised) by ABSS on a per-second basis for the entire duration of all traces in the evaluation set. All sectors held in the queue need to be copied to the remote host when migration is triggered before the VM can resume; they represent sectors that are out-of-date at the remote host.

Table VII provides the results of our simulation, including the time it would take to copy the outstanding sectors over a 1Gbps link. As shown, four out of five workloads can, on average, be completely synchronised in less than 1.1 s. MSNBE incurs the most overhead but can also be migrated in under 26 s. MSNBE has a larger working set throughout the duration of the run which is why we observe a longer latency. Additionally, the variation on the time to transfer the outstanding sectors are relatively low ranging from $\pm$50 ms in the best case to under $\pm$11 s in the worst case.

In conclusion, our results show that ABSS achieves synchronisation with reduced bandwidth requirement between 0.3% and 2.4% of 1Gbps link capacity. Additionally, The migration overhead of our algorithm is less than 1.1 s in the majority of workloads, with the worst case being only 26 s.

B. The ABSS Parameters

AS outlined in Section IV, ABSS is influenced by two tunable parameters: the required confidence interval that translated to an upper time bound (c) and the time

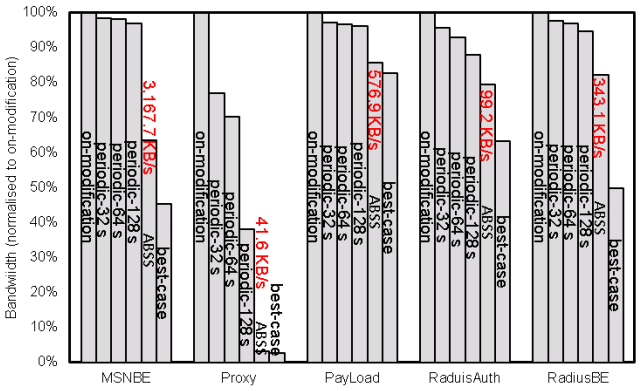

Fig. 5. Bandwidth is shown for different algorithms that are normalised to on-modification. Absolute throughput for ABSS is overlayed (red).

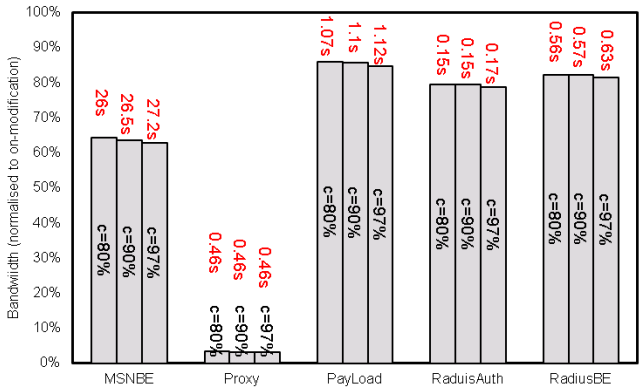

Fig. 6. Bandwidth is shown for different values of c. Avg. migration latency for a gigabit link is overlayed (red). The time threshold is set to 4,000s.

threshold after which a sector is always synchronised (`unmodTimeThreshold`). We vary these parameters in our simulations to characterise their influence.

In particular, we vary the confidence interval at 80%, 90% and 97%. As the results in Figure 6 indicate, the effects on data transfer are minimal. The average amount of data transferred is reduced by only 3% at the cost of increased migration latency up to 14% (in the case of RadiusAuth). As the Chebyshev-type bound is already pessimistic by nature, a higher value only results in holding needlessly the modified sectors longer in the queue. Consequently, setting c to 90% is better considering the bandwidth and latency trade-off.

We further analyse the effect of varying the unmodified time threshold from 500 s to 8,000 s on the amount of data transferred and the latency overhead. Figure 7 (exemplified in MSNBE) illustrates that the bandwidth requirement of ABSS is inversely related to the time threshold while we incur a proportional increase in the latency overhead. The red dots in the the figure represents the optimal threshold for each workload (we omit the whole curves for the sake of presentation). These results show that setting the threshold to 4,000 s will work reasonably well.

However, this value is essentially an application-specific tunable parameter that captures the required trade-off level between bandwidth and latency. In cases where consistency is required (e.g. for the purpose of reliability), having a small time threshold enables ABSS to be more aggressive in synchronisation but with the expense of higher bandwidth requirement. On the other hand if we can tolerate weaker consistency, setting the threshold to a large value would reduce the amount of data transferred but increase the latency overhead when migration is triggered.

C. The Case at Two Ends

We now discuss conceptually the limits of ABSS with respect to two extreme cases of write applications. We show that our proposed algorithm works well for both scenarios. We advocate that no other synchronisation algorithm is capable of achieving good results at both fronts.

The first case concerns workloads that append only to files on disk. For applications to excel at batch processing, sequential I/O is more important than random I/O. In these cases, the file system is log-structured and writes are only appended to files (e.g. Hadoop Database[6]).

As modifications are only written to the end of the file, sectors are effectively not overwritten. In other words, sectors are just written once. As our algorithm is opportunistic with respect with the first write to a sector (it assumes that most sectors are written once), the synchronisation will happen as soon as any sector is updated. Having said that, the two disk images will be identical whenever migration is needed with virtually no latency overhead incurred.

The other end of the spectrum involves workloads that just write to a few sectors; that is all sectors are continuously overwritten. In these workloads, we do not have sectors that are written once. We also have a very few sectors that are modified. As our algorithm is opportunistic, it will send the first update to a given sector. However, subsequent updates are absorbed as the sectors will be considered *hot* and retained in the queue. Consequently we do not consume any bandwidth for synchronisation while the workload writes to disk. Additionally, the outstanding sectors yet to be transferred do not result in a large migration latency as we assume that only a few sectors are updated.

VI. Implementation Considerations

This section outlines some of the major issues of consideration in a production implementation of the ABSS algorithm.

We assume that during the creation of a new VM, its disk image is replicated to possible remote destinations using templates and snapshots [13]. This one-off setup process reduces storage synchronisation to the transfer of modified disk sectors while the VM is running.

Foremost is the issue of maintaining consistent disk image replicas across multiple datacentres. While this work has focused on *identifying* candidate sectors for synchronisation, achieving synchronisation requires modified sectors be propagated to *all* datacentres to which the VM may potentially be migrated. While technologies such as multicast can help reducing the data transfer overhead of synchronisation, an

[6]http://hadoop.apache.org/

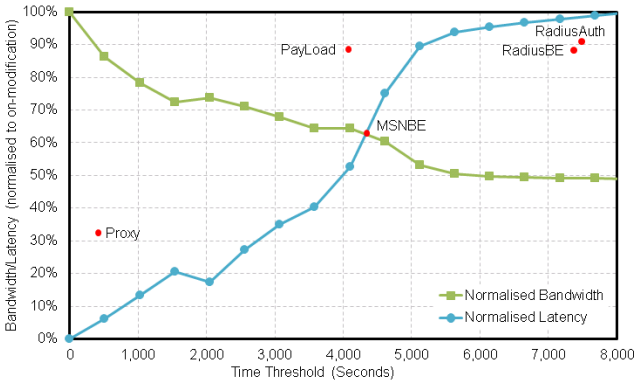

Fig. 7. ABSS bandwidth/Latency requirement for different time thresholds normalised to on-modification synchronisation. Red dots represent the intersection of the two curves.

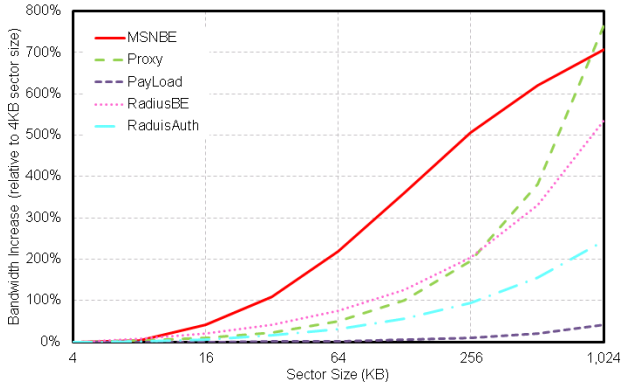

Fig. 8. Percentage of bandwidth increase of ABSS for different sector sizes relative to 4KB.

efficient implementation will require the disk synchronisation algorithm be integrated with the VM scheduling and placement layer. Similarly, support for intercepting VM write operations and block contents are necessary for the purposes of modelling and synchronising sector-state.

Additionally, a production implementation will have to incorporate constraints such as limited link capacity and contention between VM traffic with disk synchronisation traffic.

Maintaining sector-state may incur a large space overhead. For example assuming a 4KB sector size and 32-bit floating point representations, storing just the average and standard deviation for every sector in a 1TB disk image requires 2GB of storage ($\approx 0.2\%$ overhead). In a production implementation, this space overhead can be mitigated by sparse recording. Similarly, the memory overhead of the system can be limited by using stable storage as a scratch pad.

The space overhead can also be reduced by choosing larger sector sizes. For example if the sector size is 32KB, the amount of storage requirement will be 8 times smaller than 4KB as we need to keep less statistics. Obviously, the state space reduction is proportional to the size of the sector. However, this is not expected to happen for free as the bandwidth requirement for ABSS may suffer. This is mainly due to the fact that we reduce efficiency by having larger sectors because we send the entire lot even though only part of it has been modified.

Figure 8 illustrates the bandwidth increase (relative to 4KB) for different sector sizes up to 1MB. In the case of Payload, the choice of larger sector size does not influence the average bandwidth required for synchronisation. For RadiusBE, RadiusAuth and Proxy, the additional bandwidth increase can also be tolerated but for smaller sizes only. On the other hand, MSNBE incur bigger bandwidth overhead as we increase the sector size. In conclusion, the sector size choice is largely dependant on the application and its write activity behaviour but it can reduce considerably the state space.

However, implementations can be intelligent in sending only the modified portion of a sector. Techniques such as content based redundancy elimination and sub-sector deltas [14] can reduce the bandwidth overhead for having bigger sector sizes.

VII. Related Work

There is a relatively little work in the area of synchronising VM disk images across physical hosts, mainly due to the assumption that migration will only be carried out in the same datacentre as explained in Section I. The closest work we could find was by Deutsche Telekom in which the authors outline a system that copies disk-state on-migration [15]. ABSS is different from this design in that it operates while the VM is running before migration is triggered. Furthermore, it aims to reduce both the data transfer requirements and the VM migration time while this related work focuses solely on providing synchronisation functionality.

XNDB migrates memory-state to the destination host and *then* fetches required disk sectors from the source on-demand [16]. Additionally, the design of this system relegates disk synchronisation to a background process. While this architecture reduces the VM migration time, it may lead to possible service degradation due to the latency overhead of fetching remote sectors. Furthermore, the source host cannot be retired until synchronisation is complete.

Ramakrishnan et al. outline an asynchronous storage synchronisation algorithm that copies modified sectors while the VM is running [17]. When a migration is requested, it switches to synchronous mode to guarantee strong consistency. Similarly, CloudNet [14] synchronises disk-state through the use of Distributed Replicated Block Device (DRBD)[7], which operates by mirroring whole block devices. Both these systems would benefit from the integration of ABSS.

Seneca [18] is a remote storage mirroring for the purpose of disaster recovery. It uses asynchronous protocol and write coalescing to ensure high performance while minimising the amount of data transferred. Similarly, the REMUS [19] project provides a high degree of fault tolerance using asynchronous VM replication. We believe that ABSS could extend these projects by providing an adaptive algorithm at the sector level.

Finally, distributed file systems with good caching capability

[7] http://www.drbd.org/

(e.g. Lustre[8]) have been shown to work effectively for the purpose of VM migration [20]. For these systems, ABSS can provide a back-end storage with a replication mechanism based on sector level statistics. Additionally, our algorithm enables a dynamic caching policy according to the write behaviour.

VIII. Future Work

In our research into Computing for the Future of the Planet we are creating a novel platform that automatically relocates workloads to geographically distant sites with surplus renewable energy that would otherwise be wasted [21], [22]. Our platform utilises live migration for workload relocation and thus disk synchronisation is a necessary component of the proposed system. Currently we are implementing ABSS as part of this bigger architecture.

Furthermore, we are in the process of unifying our previous work on memory state migration [8] with ABSS to create one VM migration algorithm that allows operators to transparently and efficiently move VMs across WAN links without violating service level agreements.

IX. Conclusion

This paper outlined ABSS, a statistical algorithm designed to synchronise disk images of running VMs across WAN links at the sector level. ABSS aims to minimise both the amount of data transferred and the VM migration time. Based on observations drawn from production workloads, ABSS operates by creating an adaptive model of the sector write behaviour. Our statistical modelling is used to predict the upper time bound for the next write to a sector, only synchronising it if no update occurs.

ABSS is evaluated against the best-of-breed algorithms in terms of the amount of data transferred and the VM migration time. Trace-driven simulations on five production workloads show that ABSS is able to achieve useful trade-off having a data transmission overhead between 0.3% and 2.4% of a 1Gbps link whilst incurring in the majority of cases a migration latency of less than 1.1 s.

Finally, this paper also provides an analysis of the write activity of five modern workloads, providing details (at the sector level) of the temporal characteristics of disk modification.

X. Acknowledgments

We are grateful to Andrew W. Moore for his support throughout this work. We would like also to thank Stelios Timotheou and Simon Fothergill for their useful feedback.

References

[1] 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 *Proc. ACM Symposium on Operating Systems Principles (SOSP'03)*, New York, NY, USA, 2003, pp. 164–177.

[2] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'05)*, Berkeley, CA, USA, 2005, pp. 273–286.

[3] R. S. Tucker, R. Parthiban, J. Baliga, K. Hinton, R. W. A. Ayre, and W. Sorin, "Evolution of WDM Optical IP Networks: A Cost and Energy Perspective," *J. Lightwave Technol.*, vol. 27, no. 3, pp. 243–252, Feb. 2009.

[4] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *Trans. Storage*, vol. 4, pp. 10:1–10:23, November 2008.

[5] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production windows servers," in *Proc. IEEE Symposium on Workload Characterization (IISWC'08)*, October 2008, pp. 119–128.

[6] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *ACM Commun.*, vol. 51, no. 1, pp. 107–113, January 2008.

[7] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proc. ACM International Conference on Management of Data (SIGMOD'09)*, New York, NY, USA, July 2009, pp. 165–178.

[8] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the performance of virtual machine migration," *Proc. IEEE Symposium on Modeling, Analysis, and Simulation of Computer Systems (MASCOTS'10)*, vol. 0, pp. 37–46, August 2010.

[9] D. A. Patterson, "Technical perspective: the data center is the computer," *ACM Commun.*, vol. 51, no. 1, pp. 105–105, January 2008.

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

[11] J. G. Saw, M. C. K. Yang, and T. C. Mo, "Chebyshev inequality with estimated mean and variance," *The American Statistician*, vol. 38, no. 2, pp. pp. 130–132, May 1984.

[12] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. pp. 419–420, August 1962.

[13] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 377–390, December 2002.

[14] 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 *Proc. ACM Conference on Virtual Execution Environments (VEE'11)*, New York, NY, USA, March 2011, pp. 121–132.

[15] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *Proc. ACM Conference on Virtual Execution Environments (VEE'07)*, New York, NY, USA, June 2007, pp. 169–179.

[16] T. Hirofuchi, H. Ogawa, H. Nakada, S. Itoh, and S. Sekiguchi, "A Live Storage Migration Mechanism over WAN for Relocatable Virtual Machine Services on Clouds," in *Proc. IEEE/ACM Symposium on Cluster Computing and the Grid (CCGRID'09)*, Washington, DC, USA, 2009, pp. 460–465.

[17] K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "Live data center migration across WANs: a robust cooperative context aware approach," in *Proc. ACM Workshop on Internet Network Management (INM'07)*, New York, NY, USA, August 2007, pp. 262–267.

[18] A. V. Minwen Ji and J. Wilkes, "Seneca: Remote mirroring done write," in *Proc. USENIX Annual Technical Conference, General Track*, Berkeley, CA, USA, June 2003, pp. 253–268.

[19] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "REMUS: high availability via asynchronous virtual machine replication," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*, Berkeley, CA, USA, 2008, pp. 161–174.

[20] A. Ortiz, F. Thiebolt, P. Stolf, G. D. Costa, and A. Sayah, "Virtual machine migration: A comparative study of storage viewpoints," in *Proc. Civil-Comp International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering (PARENG'11)*, Stirlingshire, UK, April 20011.

[21] A. Hopper and A. Rice, "Computing for the future of the planet," *Philosophical Transactions of the Royal Society*, vol. 366, no. 1881, pp. 3685–3697, 2008.

[22] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Free lunch: Exploiting renewable energy for computing," in *Proc. USENIX Workshop on Hot Topics in Operating Systems (HotOS'11)*, Berkeley, CA, USA, May 2011.

[8] www.lustre.org/