TideWatch: Fingerprinting the Cyclicality of Big Data Workloads

Dan Williams* Shuai Zheng‡ Xiangliang Zhang‡ Hani Jamjoom*

*IBM T. J. Watson Research Center, Yorktown Heights, NY
‡King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia

***Abstract*—Intrinsic to "big data" processing workloads (e.g., iterative MapReduce, Pregel, etc.) are cyclical resource utilization patterns that are highly synchronized across different resource types as well as the workers in a cluster. In Infrastructure as a Service settings, cloud providers do not exploit this characteristic to better manage VMs because they view VMs as "black boxes." We present *TideWatch*, a system that automatically identifies cyclicality and similarity in running VMs. TideWatch predicts period lengths of most VMs in Hadoop workloads within 9% of actual iteration boundaries and successfully classifies up to 95% of running VMs as participating in the appropriate Hadoop cluster. Furthermore, we show how TideWatch can be used to improve the timing of VM migrations, reducing both migration time and network impact by over 50% when compared to a random approach.**

I. Introduction

Massively parallel data processing workloads, such as MapReduce [8] and Hadoop [2], are designed for scale. They are commonly deployed on Infrastructure as a Service (IaaS) clouds as part of a growing trend towards "big data." By design, these workloads are iterative or cyclical in nature. A typical Hadoop workload, for example, consists of a number of VMs. In each iteration, all VMs[1] executes similar MapReduce operations that process different chunks of the underlying data. In cloud deployments, however, IaaS providers do not typically differentiate between workloads, mostly because they treat VMs as black boxes. In this paper, we focus on the following two questions: *(1) how cyclical are Hadoop workloads, and (2) how can cyclicality be efficiently detected and exploited by a cloud provider without a priori knowledge about the running workload?*

There is a number of studies that have looked at VM workload characterization and placement [21], [30], [27], [34], [35], to name a few. Unlike previous work, we closely examine the cyclicality of Hadoop workloads from the perspective of a cloud provider. We analyze three popular data mining operators (k-means, Dirichlet Process Clustering, and PageRank) on a local Hadoop testbed and on Amazon EC2. We confirm that Hadoop workloads are cyclical across CPU, memory, and network. In particular, slave VMs experience periods of heavy I/O (to read, write, or shuffle the data) followed by high CPU utilization (to process the data). More importantly, cyclicality within the same Hadoop cluster is highly synchronized across all resources and across VMs in the same cluster.

[1]In reality, a master VM is used for coordinating the execution of slave VMs

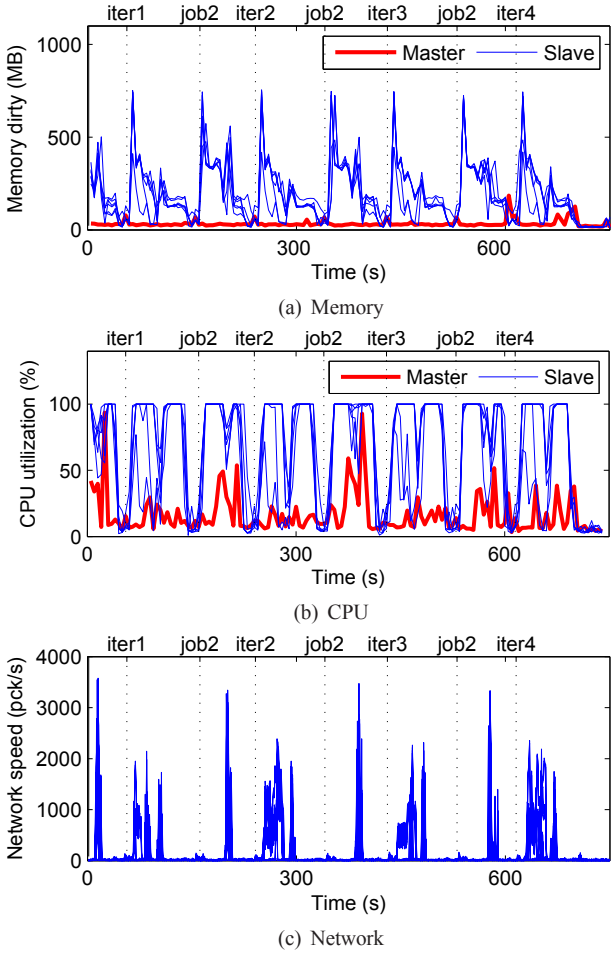
We present *TideWatch*, a system that enables a cloud provider to automatically fingerprints the cyclicality and similarity within VMs, without a priori knowledge about the workload inside the VM. TideWatch monitors resource utilization, then detects the cyclicality of each VM in real time by converting the utilization data into a square wave—from which period length is computed. TideWatch also identifies groups of VMs that are part of the same workload using dynamic time warping (DTW) [6] to compute similarity between VM resource utilization patterns and density-based clustering with noise (DBSCAN) [11] to group them.

TideWatch can be applied to optimize VM management techniques, including scheduling and placement. In particular, by detecting the valleys within a workload cycle, TideWatch can minimize the amount of (writable working set) memory that needs to be transferred between physical hosts. We have deployed TideWatch at the hypervisor-level of a local testbed and have also used TideWatch to analyze data from Hadoop deployments on Amazon EC2. TideWatch predicts period lengths of most VMs in Hadoop workloads within 9% of actual iteration boundaries and successfully classifies up to 95% of running VMs as participating in the appropriate Hadoop cluster. When applied to live VM migration, TideWatch reduced migration time by 53% and reduced network cost by 52%, when compared to a random approach. TideWatch also enabled more predictable migration costs: standard deviation is reduced by a factor of 15.

In this paper, we make the following contributions: (1) we confirm the existence of resource utilization cyclicality in iterative Hadoop workloads, (2) we present mechanisms to automatically detect the existence and period length of resource utilization cycles in VMs, (3) we describe techniques to automatically group VMs operating on similar workloads, and (4) we show how cycles and VM groups can be applied to optimize VM management. All of the above is achieved while treating VMs as a black boxes.

The rest of this paper is organized as follows. Section II describes the experimental setup. Section III examines cyclicality in Hadoop data mining algorithms. Section IV presents TideWatch, and how it detects cyclicality, estimates period lengths, and identifies Hadoop clusters. Section V discusses applications of TideWatch to optimize VM management. Finally, Section VI evaluates TideWatch, Section VII surveys related work, and Section VIII concludes.

978-14799-3360-0/14/$31.00 ©2014 IEEE

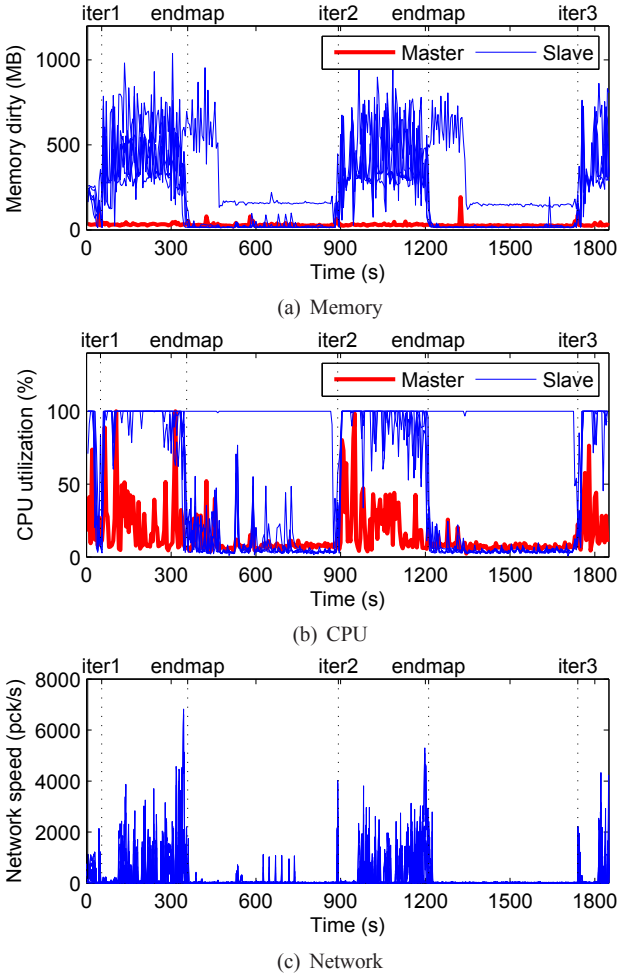


(a) Memory

(b) CPU

(c) Network

Fig. 1. PageRank memory, CPU and network activities (on local testbed)



(a) Memory

(b) CPU

(c) Network

Fig. 2. Dirichlet memory, CPU and network activities (on local testbed)

II. Methodology

We evaluate three popular data mining algorithms in Apache Hadoop.

PageRank. PageRank [23] measures the importance of a node in a graph by finding the principal eigenvector of the hyperlink matrix defined by links among nodes. We use PEGASUS [17] to run PageRank on top of Hadoop, which computes the importance vector by running two MapReduce jobs in each iteration.

Dirichlet process clustering. Dirichlet is a model-based clustering algorithm that identifies the assignment of objects and component parameters by sampling to maximize the posterior probability of mixture models in iterations.

K-Means. K-means is a well known iterative refinement algorithm for clustering objects into a user-specified number (k) of groups.[2]

We measure the resource utilization (sampled every 5 seconds) of three resources: memory, CPU, and network. Additionally, we apply exponential smoothing (with a smoothing factor of $\alpha = 0.15$) on the raw data to reduce the effects of noise. We use the Linux command `sar` in the guest machine to collect CPU and network utilization every 5 seconds.[3] We also capture—on a page granularity—how much memory is actively being written to during each sampling period. This metric, called the *writable working set* (WWS) [7], is a version of the working set concept [9] that has been utilized by various memory-related VM management tasks, including live VM migration [7]. To calculate the WWS periodically, we use Xen's shadow page tables [4], [7]. In particular, we periodically read (from the hypervisor) statistics detailing how many pages the VM has written.

We examine the algorithms on both a small local testbed (two Dell Precision T7500 workstations, with every VM assigned 1 VCPU, 1.7 GB memory, and a 20 GB disk image) and Amazon Elastic Compute Cloud (EC2). For consistency with the local testbed, the experiment on EC2 uses small instances.[4]

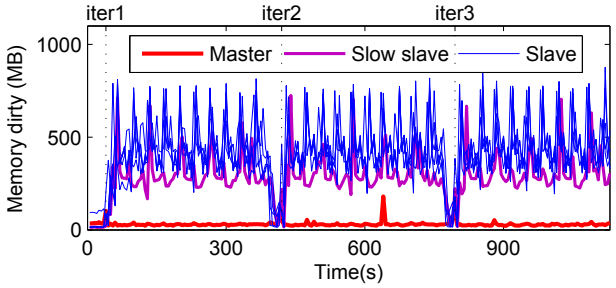
For PageRank, on the local testbed, we use a 268 MB US Patent citation network dataset[5] (with 3,774,768 nodes and 16,518,948 edges). On EC2, we run PageRank on a 1.1 GB

[2]We use the implementation from Apache Mahout (http://mahout.apache.org/)

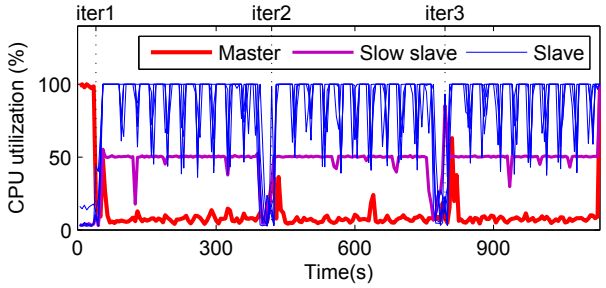
[3]Hypervisor-level monitoring (e.g., `xentop`) is also possible.

[4]Each small instance has 1 EC2 Compute Unit (1 virtual core with 1.7 GB memory). We cannot measure the writable working set (WWS) on EC2.

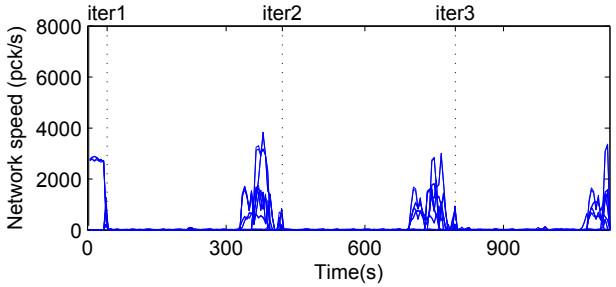
[5]http://snap.stanford.edu/data/cit-Patents.html



(a) Memory



(b) CPU



(c) Network

Fig. 3. k-means memory, CPU and network activities with one VM slowed down (on local testbed)

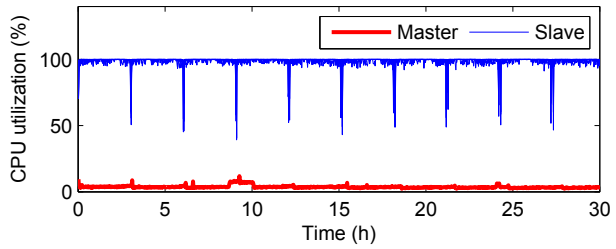


Fig. 4. CPU utilization for k-means on EC2

LiveJournal social network dataset[6] (with 4,847,571 nodes and 68,993,773 edges). We apply k-means and Dirichlet to cluster the 1.3 GB 1-gram English One Million dataset on the local testbed, and the 25 GB 2-gram English One Million dataset[7] on EC2.

III. Workload Cyclicality

We are interested in evaluating the cyclicality and synchronization of the workload across various computing resources and across worker VMs. Figure 1, 2 and 3 show the memory dirty count, the CPU utilization and network activity of VMs in our local cluster running PageRank, Dirichlet, and k-means clustering, respectively. The starting point of each iteration and map phase is delimited with an *iter* label.[8] Despite variation and potential cyclicality within an iteration, memory and CPU show a clear local minimum in the utilization at iteration boundaries.

Figure 4 shows the CPU utilization on Amazon EC2, running k-means on the larger dataset. Here, we are limited to monitoring information within the VM. The figure confirms that CPU utilization exhibits cyclicality that matches the iterations.

More importantly, we are also interested in the robustness of the synchronization across workers. We slowed down the CPU of one VM by 50% (using Xen's processor capping feature [4]) to simulate resource contention. Figure 3 demonstrates how all VMs synchronize to follow the iteration boundaries of the slowest VM.[9]

Overall, this cyclical and highly synchronized behavior matches the intuition behind the design of iterative data mining frameworks, like Hadoop. From a cloud provider's perspective, it creates an opportunity to transparently detect these workload and optimize certain cloud management operations.

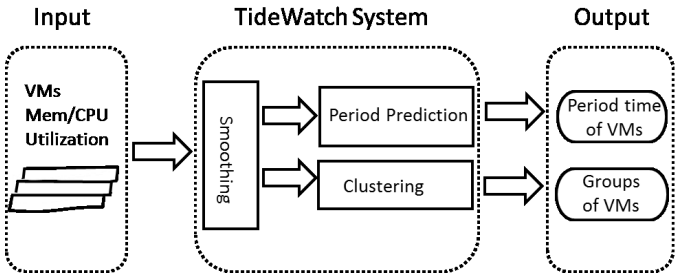


Fig. 5. TideWatch architecture

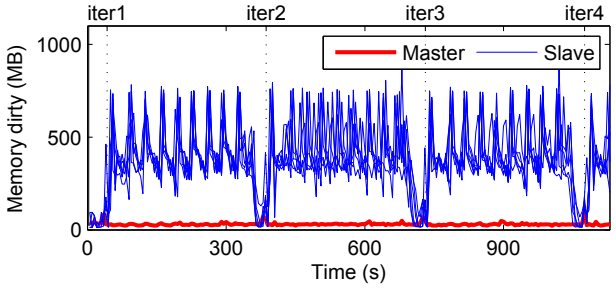
IV. TideWatch

We present TideWatch, a system that detects cyclicality in VM workloads while maintaining a black-box approach. TideWatch can also detect groups of VMs that are likely part of the same Hadoop cluster. The architecture of TideWatch is shown in Figure 5.

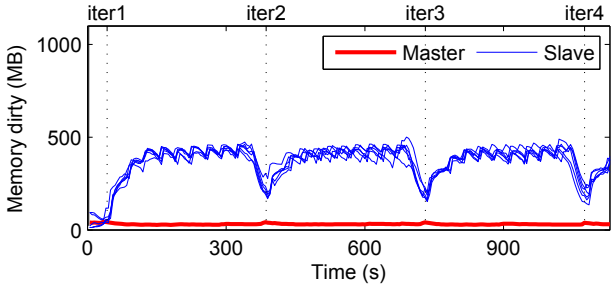
The input to TideWatch is in the form of resource utilization data. Each hypervisor gathers resource utilization data from VMs using the techniques discussed in Section II. In particular, each hypervisor tracks the memory (writable working set), CPU and network utilization, and sends them to TideWatch. TideWatch is a logically centralized entity. One instance of TideWatch can be run per machine, per rack, per pod (multiple racks) or per data center. In this section, we assume that distributed systems challenges in providing data

[6]http://snap.stanford.edu/data/soc-LiveJournal1.html
[7]http://books.google.com/ngrams/datasets

[8]The master VM follows a different utilization pattern, but may also be cyclic and share the cycle period.
[9]Without any slowdown, all VMs have a cycle of 345 seconds (not shown in the figure). The introduction of a slow VM increased the period for all VMs to 380s.



(a) Memory dirty count of k-means before exponential smoothing



(b) Memory dirty count of k-means after exponential smoothing

Fig. 6. Example of exponential smoothing

to TideWatch have been met and instead focus on processing the resource utilization data.

As shown in Figure 5, TideWatch first applies exponential smoothing [14] to remove noise from the raw resource utilization data (see Figure 6). The smoothed data is processed in two ways. First, *period prediction* identifies cycle periods in each VM. This is accomplished using digital signal processing. Second, *clustering* identifies VMs that belong to the same Hadoop workload. This is accomplished through a combination of Dynamic Time Warping (DTW) [6] and density-based clustering with noise (DBSCAN) [11].

The output of TideWatch is a period duration estimation for each VM and a grouping of VMs that are suspected to be working on the same Hadoop workload. In Section V, we apply this output to minimize the impact of management tasks, such as live VM migration, on the underlying infrastructure. The remainder of this section describes the analyses performed by TideWatch in more detail.

A. Period Prediction

On each VM, TideWatch estimates—in real-time—the period of resource utilization cyclicality. As shown in Section III, iteration boundaries involve the biggest change in resource utilization. For example, both memory and CPU measurements show their lowest values at iteration boundaries. In order to distinguish iteration boundaries from smaller variation in resource utilization, TideWatch uses a threshold to further smooth out the data. As shown in Figure 7, it digitizes the time series X by computing a threshold T. We found that setting the threshold as the midway point is both robust and sufficient. Specifically,

$$T = X_{min} + \frac{X_{max} - X_{min}}{2},$$

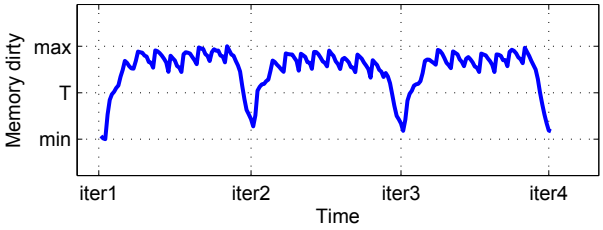


Fig. 7. Example of finding cycle period

where X_{max} and X_{min} are the maximum and minimum value of time series X, respectively. The time series X is then converted into a square wave SW with values 1 and 0.

$$SW(i) = \begin{cases} 0, & \text{if } X(i) \leq T; \\ 1, & \text{otherwise.} \end{cases}$$

The resulting square wave has the same period—aligned on iteration boundaries—as the Hadoop slaves that contributed the raw data.

The square wave SW is segmented into $\{sw_i\}$, which consists of the duration between $0 \rightarrow 1$ transitions. The estimated period time of X is determined by the median length of segments $\{sw_i\}$. By computing the median, noise which was not removed with exponential smoothing is effectively ignored.

As a side-effect of this digital signal processing, TideWatch can also calculate the duty cycle, which is the ratio of *ON* status during a period [29]. In our work, the *ON* state duration is the time when the square wave SW has value 1. The duty cycle

$$\delta = \frac{length(SW = 1)}{length(SW = 1) + length(SW = 0)}$$

measures the occupation rate of one workload on one resource. As discussed further in Section V, the duty cycle can be used to inform VM placement decisions.

B. Clustering

TideWatch groups VMs that have similar resource utilization patterns. For Hadoop workloads, these groups correspond to slave VMs that are operating as part of the same Hadoop cluster. Grouping VMs together allows TideWatch to gather more sample points and increase confidence in observed resource utilization patterns.

Despite the application of exponential smoothing, there is likely to be some variation in the resource utilization data from two VMs even if they are part of the same Hadoop cluster. However, all VMs in the same Hadoop cluster will continue to share the same basic cyclical patterns and period length as one another even if the period duration changes due to resource contention or a decrease in the number of slave machines. TideWatch uses DTW [6], to measure the distance between two time series that may vary in length and in evolving speed but have the same cyclical behavior.

DTW compares two time series in certain non-linear variations in the time dimension. It computes the distance between

two time series X (of length M) and Y (of length N) based on a "local match" score matrix [12]

$$S(i,j) = d(X(i), Y(j)),\ 0 \le i \le M, 0 \le j \le N,$$

where $d(x,y)$ can be any difference measure, e.g., Manhattan distance (absolute value of the difference). This $M \times N$ matrix S evaluates the local cost measure for each pair of elements of the sequences X and Y. Then, dynamic programming is used to find an alignment between X and Y having minimal overall cost. An accumulated cost matrix D is defined as

$$D(i,j) = S(i,j) + min\{D(i-1,j), D(i,j-1), D(i-1,j-1)\},$$
$$0 \le i \le M, 0 \le j \le N,$$

Finally, the DTW distance of X and Y is [18]

$$DTW(X,Y) = D(M,N).$$

After computing the DTW distance of each pair of VMs [10], [31], TideWatch employs DBSCAN [11], a density-based clustering algorithm, to identify groups of VMs that are working on the same Hadoop job. DBSCAN does not require TideWatch to specify the desired number of clusters, which is difficult to know in advance. Given the DTW distance measure among VMs, DBSCAN will gather VMs having the lowest distance to each other in one group and predict that they are working on the same workload. Two parameters of DBSCAN, neighborhood radius ϵ and minimal number of neighbors $N_{neighbor}$, are set to be $\epsilon = 0.1$ and $N_{neighbor} = 2$ since the minimal number of slave nodes is 2 and 90% of VMs have distance less than 0.1 to their 2nd nearest neighbors. DBSCAN also reports *outliers*, which are very different from and cannot be grouped into any clusters. In a Hadoop cluster, master nodes can behave differently from slave nodes and therefore may be classified as outliers.

The process of detecting clusters is $\mathcal{O}(n^2)$, where n is the number of VMs being considered. Depending on the size of n, cluster detection may be run frequently with low overhead. For example, clustering can be limited to the VMs instantiated by a single cloud user. Clustering of 240 VMs can be completed within 13 seconds. If clustering needs to be run more frequently, an online clustering algorithm may be used [5].

V. TideWatch for Optimizing VM Management

In this section, we describe how a cloud provider can use the output of TideWatch to optimize the live migration and placement of VMs.

A. Live VM Migration

Live VM migration is a VM management feature that is available on all major virtualization platforms [7], [22], [20]. It enables a running VM to be moved to another physical machine with virtually no downtime for reasons of maintenance [7] or balancing load across physical machines [35], [33].

While live VM migration has little impact on the performance of the migrating VMs, it is an operation that generates a large amount of traffic on the cloud provider's network. A migration operation can also be slow to take effect due to long total migration time, which limits its applicability for load balancing [33]. The hypervisor executes live VM migration by transmitting the memory pages allocated to the migrating VM across the network while the VM is still running. Importantly, the hypervisor tracks which pages have been modified since they were transmitted and retransmits those pages. This process proceeds in iterations until the set of modified pages are small enough for the VM to be suspended at the source and resumed at the target.

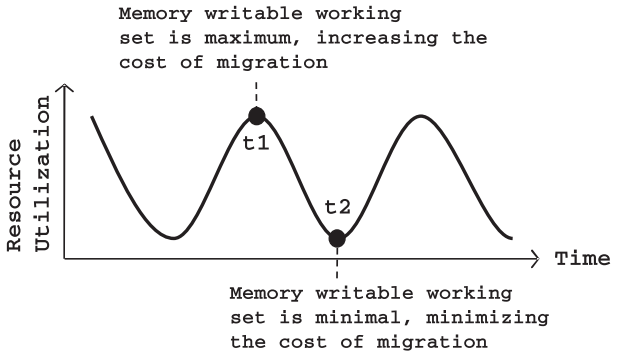


Fig. 8. Effects of memory working set on migration cost

The amount of network traffic generated and total migration time of a live migration operation are a function of the amount of memory allocated to the VM, and the rate at which the VM modifies (dirties) pages. Figure 8 shows how the cyclicality of memory utilization (specifically the writable working set described in Section II) exposed by TideWatch can be used to schedule migration operations. When the writable working set is minimal ($t2$ in Figure 8), less memory must be transferred resulting in a reduction in network traffic and migration duration.

B. VM Placement

The cloud provider can select where in a data center each VM is placed. It has the flexibility to use VM placement to optimize a number of aspects of the data center, including overall utilization, utilization of a specific shared resource like the network, or the performance of VMs [35], [19], [36]. In this subsection, we discuss how the information exposed by TideWatch complements or informs placement strategies.

The cyclical resource utilization behavior of VMs identified by TideWatch can be used to predict utilization and reduce overprovisioning. Overprovisioning is a common technique used by cloud providers in which VMs are allotted more resources than they actually need, just in case a resource utilization spike occurs. By recording the peak CPU or network utilization value in previous cycles, a cloud provider can improve hotspot prediction for any placement strategy (e.g., [35], [33]).

TideWatch also creates opportunities to align workloads that share resources on a physical machine or a network segment, improving on [21], [27]. For example, consider workloads with low duty cycles for a particular resource, such as network utilization for k-means (Figure 3(c)). By co-locating this workload with a similar workload that uses the

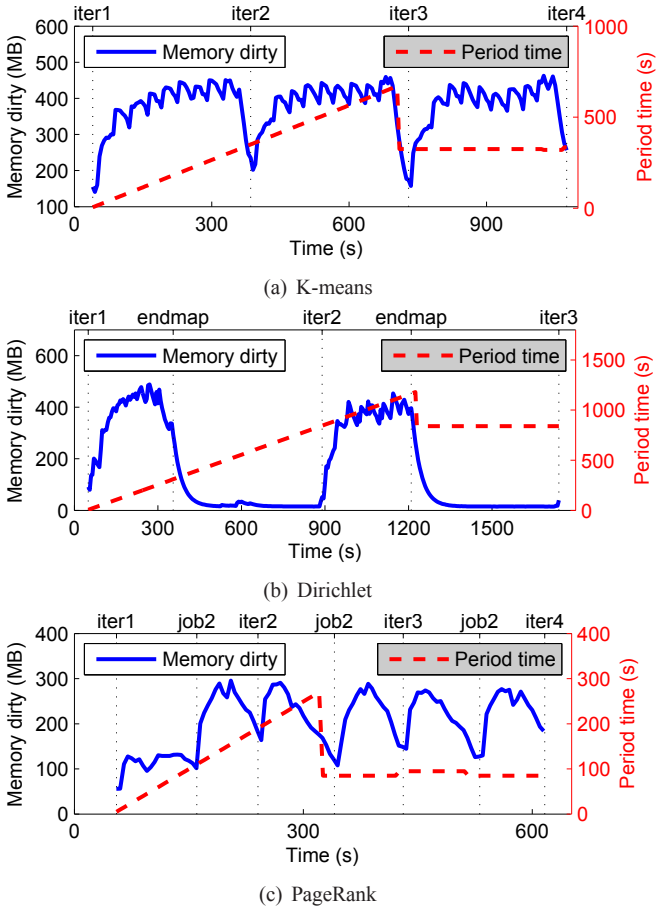


Fig. 9. Period predicted in real-time

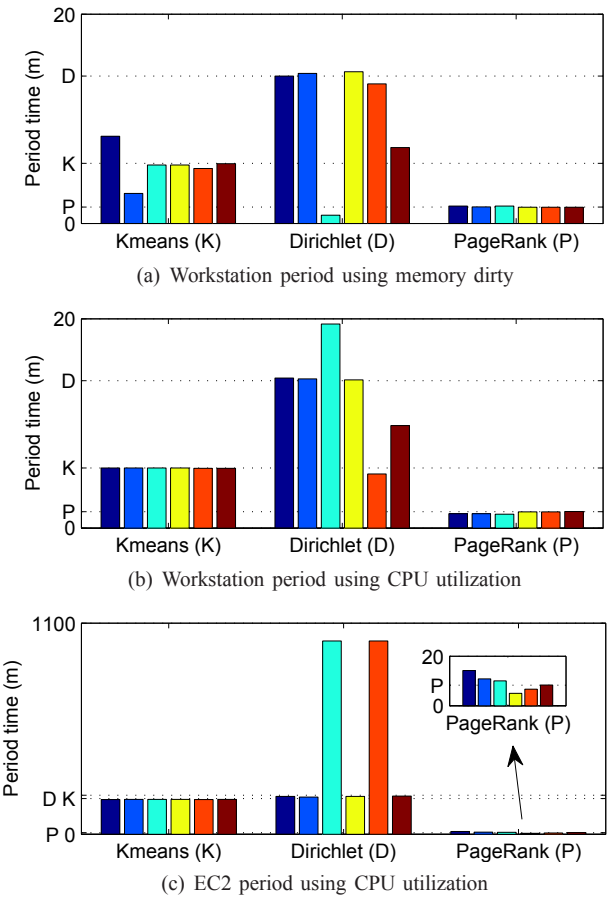


Fig. 10. TideWatch period for each VM

network at complementary times, network utilization would be less bursty, with less contention.

Finally, TideWatch can help reduce the effect that "noisy neighbor" VMs [24], or VMs that create contention on shared resources like the network, have on Hadoop workloads. VMs in an iterative Hadoop workload operate in synchrony with each other. If one VM is slowed down (e.g., due to contention with a noisy neighbor), all slave VMs may be stalled before they can continue with subsequent iterations. By detecting VMs that are in the same cluster, TideWatch enables a cloud provider to calculate the number of VMs that may be affected and then either avoid or rectify detrimental placement decisions.

VI. Evaluation

In this section, we evaluate TideWatch—both period prediction and clustering—and its application to live VM migration. For consistency, we use the monitored resource utilization data gathered in Section III as input to TideWatch. For all experiments, TideWatch first applies exponential smoothing (with $\alpha = 0.15$) on the raw data to reduce the noise in the collected data.

A. Period Prediction

To evaluate the period prediction aspects of TideWatch, we first examine how much data TideWatch requires before the period prediction converges. Figure 9 shows the smoothed memory dirty rate (blue curves) of one slave VM from each Hadoop workload running on the local testbed, and the corresponding predicted periods (red curves) by TideWatch in real time. The value of the period predicted at each point in time is shown on the right y-axis. At the start of the experiment, the predicted period increases linearly because TideWatch does not find multiple $0 \rightarrow 1$ transitions in the square wave (Section IV-A). In all cases, after approximately two iterations, the period estimate converges.

Figure 10 shows the values that the period prediction converges to after processing the data stream (only for slaves). TideWatch calculates the period time for each VM independently. The six bars in every group correspond to the period time estimated by TideWatch for all six slave VMs in each workload. The actual period time reported in the Hadoop logs, which represents a "ground truth" that is not readily accessible to the cloud provider, is labeled on the y-axis, with "K" for k-means, "D" for Dirichlet, and "P" for PageRank. For k-means in Figure 10(a), the period predicted of 4 VMs out 6 have less than 9% error. For Dirichlet in Figure 10(a), 4 out of 6 of the predicted period have less than 6% error. For PageRank in Figure 10(a), all the VMs have less than 8% error, 4 of them have less than 2%. The larger prediction errors for some VMs in the k-means and Dirichlet workloads stem from noise in the workload that prevents the threshold to be crossed at an iteration boundary or causes the threshold to be crossed at another time. We expect the prediction to

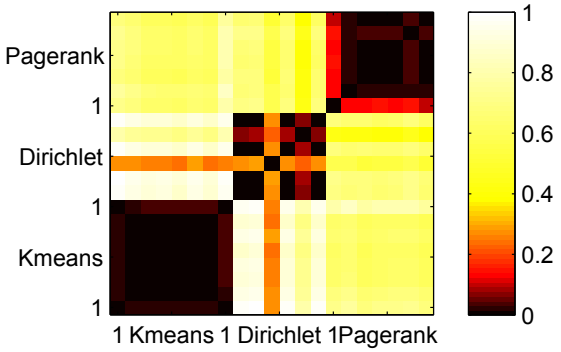


Fig. 11. Local testbed DTW distance (memory)

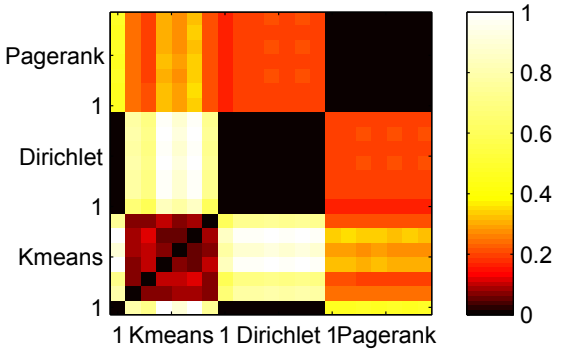


Fig. 12. Amazon EC2 DTW distance (CPU)

improve significantly with more iterations in the time series, because using the median prediction should eliminate outliers. Using CPU utilization to predict period length, shown in Figure 10(b) and 10(c), similarly yields good results for k-means and PageRank. TideWatch has good results for three (50%) of the Dirichlet VMs in Figure 10(b) and four (66.7%) of the Dirichlet VMs in Figure 10(c).

B. Clustering

TideWatch uses DTW and DBSCAN to detect groups of VMs running the same Hadoop workloads. In this subsection, we evaluate the efficacy of DTW as a distance measure and how well VMs are clustered with DBSCAN. We also evaluate the performance of detecting clusters and the robustness of Hadoop clusters in the face of heterogeneous resource contention.

Figure 11 shows the DTW distance of local testbed experiments using the memory utilization data. Figure 12 shows the DTW distance of Amazon EC2 experiments using the CPU utilization data. Each workload consists of 7 VMs: one master and six slaves. Both figures show a 21×21 grid, where the x- and y-axes correspond to unique VM ids. The DTW distance of VM_i and VM_j is indicated by a color value in cell(i, j). In Figure 11, the first node (labeled 1) on the x- and y-axes is the master node of the k-means Hadoop workload, the next six are the slave nodes of k-means Hadoop workload; similarly, the remaining values correspond to VMs in the Dirichlet process clustering workload and the PageRank workload.

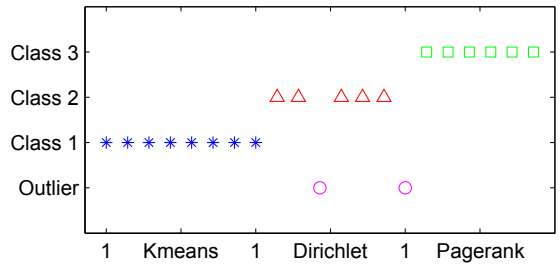


Fig. 13. Local testbed DBSCAN result (memory)

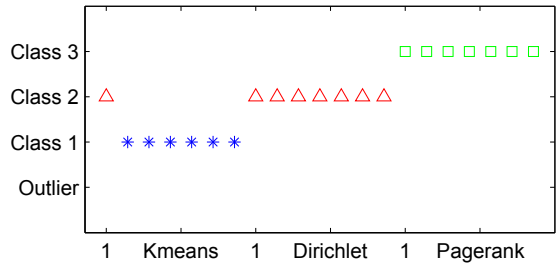


Fig. 14. Amazon EC2 DBSCAN result (CPU)

The figures show that the slave nodes of the same Hadoop workload have low DTW distance with each other, corresponding to a high degree of similarity (depicted by the three dark boxes along the $y = x$ diagonal). On the local testbed (Figure 11), there is one slave node of Dirichlet with higher DTW distance with other slave nodes of the same cluster, because it was configured to perform reduce jobs. Furthermore, the master node of each Hadoop workload usually has high DTW distance compared to the DTW distance between the slaves. In particular, the master nodes of Dirichlet and PageRank have higher DTW distance with slave nodes in the same clusters. On Amazon EC2 (Figure 12), similarly, the slave nodes of the same workload have low DTW distance with each other.

Figures 13 and 14 show DBSCAN clustering results using the DTW distances on the local testbed and Amazon EC2 respectively. The x-axes consist of the 21 VM ids, in the same order as Figures 11 and 12. The y-axes show the clusters that result from DBSCAN, as well as outliers. On the local testbed (Figure 13), only two of the 21 VMs are misclassified. The master node of Dirichlet is misclassified into the k-means cluster, and one slave node of Dirichlet (the slave configured to perform reduce tasks) is misclassified as an outlier. It is not an error that the master node of the PageRank cluster is classified as an outlier, since master nodes can behave differently from slave nodes in one Hadoop workload. On Amazon EC2 (Figure 14), all of the VMs were classified correctly except for the master node of k-means.

In Figure 15, we show the computation time for DTW and DBSCAN as the number of VMs increases. To generate this plot, we represent each VM by a time series made from 100 random numbers. This time series can be memory dirty count (like the local testbed experiments) or CPU utilization (like the EC2 experiments). The performance of DTW and DBSCAN is shown in the figure for clustering various numbers of VMs, and fit with a quadratic polynomial. If TideWatch limits

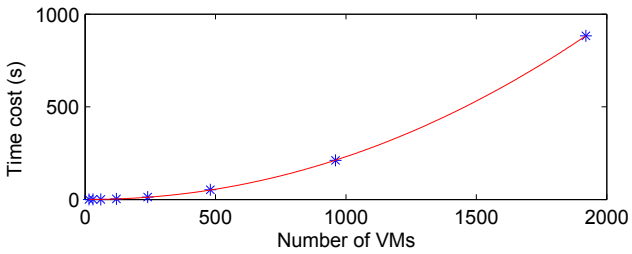


Fig. 15. DTW and DBSCAN time cost

TABLE I. AVERAGE TOTAL MIGRATION TIME AND NETWORK COST [WITH STANDARD DEVIATION]

	Total time (s)	Data transmitted (GB)
Random	33.68 [8.80]	2.80 [0.73]
TideWatch	21.84 [0.56]	1.85 [0.04]

clustering—e.g., to VMs that all belong to an individual cloud user account—it will perform well. For example, TideWatch can cluster 120 VMs in 3.2 seconds.

As seen in Section III, slaves in a Hadoop cluster tend to be synchronized in their cyclicality. This suggests that, by identifying the Hadoop cluster, a cloud provider can calculate which VMs may be affected by resource contention, for example, from a noisy neighbor.

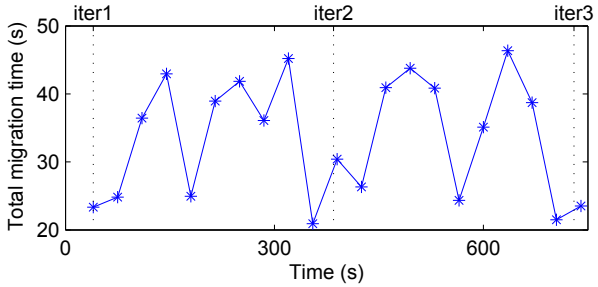
C. Live VM Migration

In this subsection, we evaluate the benefits TideWatch brings to live VM migration. On the local testbed, we run the k-means workload (Section III) and live migrate one of the slave VMs between workstations. Figures 16(a) and 16(b) show the migration time and data transmitted over the network during the migration, respectively, for migration operations triggered at different moments in time. Migrating a VM around the iteration boundary is relatively inexpensive. This is because, as shown in Figure 3(a), memory is modified at a slower rate around the iteration boundary, leading to fewer memory copy iterations during live migration.

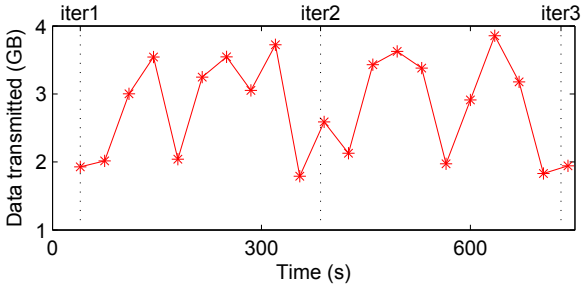
Table I shows the total migration time and network cost performance for two migration strategies, random and informed by TideWatch. The results are averaged over 21 migrations. On average, TideWatch outperforms the random strategy by reducing total migration time by 35% and data transmitted by 34%. The random strategy also has much higher variability, as evidenced by a factor of 15 higher standard deviation. For the random strategy, total migration time varies between 20.91s and 46.36s, while the traffic generated on the network varies between 1.79 GB and 3.85 GB. In the worst case, TideWatch can reduce total migration time by up to 53% and data transmitted by up to 52%.

VII. RELATED WORK

TideWatch detects Hadoop [2] clusters and the cyclicality of their resource utilization in cloud computing environments, while treating VMs as black boxes. As data-intensive computations, like Hadoop workloads, become more popular in



(a) Total migration time



(b) Data transmitted on network

Fig. 16. Costs of migration

these environments [13], some cloud providers are offering new interfaces to cloud users that enable them to identify such workloads and manage them efficiently. For example, Amazon Web Services [1] provides a service called "Elastic MapReduce," where users can submit Hadoop jobs directly. TideWatch, on the other hand, identifies these workloads for efficient management while maintaining the standard VM interface, enabling users to run customized Hadoop configurations or implementations.

We apply TideWatch to optimize the scheduling of live VM migration. VM migration across distinct physical hosts is a popular VM management technique. Although "non-live" migration techniques exist [25], [26], live migration [7], [22], [16], which has little impact on the running applications, is more popular, with implementations in most hypervisors (Xen [7], VMware [22], and KVM [20]). Live migration can be either *pre-copy*, in which memory is sent in iterations before the VM is suspended and resumed on the target, or *post-copy*, in which memory is fetched after the suspend and resume operations. TideWatch optimizes the more popular pre-copy live migration, by ensuring the number of iterations (and the number of pages copied in each iteration) is minimal.

Other systems schedule VM migration operations based on utilization. Wood et al. [35] use migration to alleviate hotspots in consolidated data centers. In an effort to limit the impact of VM migrations, Andreolini et al. [3] use a trend analysis instead of triggering migration when machine utilization passes a threshold. Stage and Setzer [28] advocate long-term migration plans involving migrations of varying priority in order to avoid network link saturation. By using TideWatch, a cloud provider can not only reduce the impact of each migration operation, but minimize the migration time, thereby enabling a cloud provider to better respond to hotspots.

TideWatch enables opportunities for informed VM placement. Many existing systems attempt to optimize VM place-

ment to achieve a various objective. Khanna et al. [19] use heuristics to try to consolidate VMs on the fewest physical machines possible. Hermenier et al. [15] and Van et al. [32] attempt to consolidate VMs while minimizing the number of VM migrations to achieve the placement objective. Predictable cyclicality implies that VMs can be packed to fully utilize physical resources, without needing to constantly adjust VM placement with migration operations.

VIII. Conclusion

We present TideWatch, a system that enables a cloud provider to monitor and predict the cyclicality of cloud workloads, while maintaining a "black-box" approach. The output of TideWatch, in particular the period duration and grouping of VMs, can be exploited to optimize VM management in a number of dimensions. To date, we have applied TideWatch to the scheduling of live VM migration operations, with promising results. TideWatch enables up to a 53% reduction in migration time and up to 52% reduction in network cost—with significantly lower variation—when compared to a random approach. As TideWatch is applied to optimize an increasing set of VM management tasks, we expect cloud providers to be able to increase the quality of their service and lower their costs, without requiring explicit cooperation from cloud users.

References

[1] "Amazon Web Services," http://aws.amazon.com/ec2/.

[2] "Hadoop MapReduce Programming Model, Documentation and Implementation," http://hadoop.apache.org/mapreduce.

[3] M. Andreolini, S. Casolari, M. Colajanni, and M. Messori, "Dynamic load management of virtual machines in a cloud architecture," in *Proc. of ICST CLOUDCOMP*, Munich, Germany, Oct. 2009.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Proc. of ACM SOSP*, Bolton Landing, NY, Oct. 2003.

[5] J. Beringer and E. Hüllermeier, "Online clustering of parallel data streams," *Data & Knowledge Engineering*, vol. 58, no. 2, pp. 180–204, 2006.

[6] D. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *AAAI-94 workshop on knowledge discovery in databases*, vol. 2, 1994.

[7] 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. of USENIX NSDI*, Boston, MA, May 2005.

[8] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[9] P. Denning, "Working sets past and present," *Software Engineering, IEEE Transactions on*, vol. 6, no. 1, pp. 64–84, 1980.

[10] D. Ellis, "Dynamic time warp (dtw) in matlab," http://labrosa.ee.columbia.edu/matlab/dtw/.

[11] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data mining*, vol. 1996. AAAI Press, 1996, pp. 226–231.

[12] J. Foote *et al.*, "Methods for the automatic analysis of music and audio," in *In Multimedia Systems*. Citeseer, 1999.

[13] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, "Statistics-driven workload modeling for the cloud," in *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*. IEEE, 2010, pp. 87–92.

[14] E. Gardner Jr, "Exponential smoothing: The state of the art," *Journal of Forecasting*, vol. 4, no. 1, pp. 1–28, 1985.

[15] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proc. of ACM VEE*, Washington, DC, Mar. 2009.

[16] M. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *Proc. of ACM VEE*, Washington, DC, Mar. 2009.

[17] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations," in *Proc. of IEEE ICDM*, 2009, pp. 229–238.

[18] E. Keogh and C. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowledge and information systems*, vol. 7, no. 3, pp. 358–386, 2005.

[19] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performace management in virtualized server environments," in *Proc. of IEEE/IFIP NOMS*, Vancouver, Canada, Apr. 2006.

[20] A. Kivity, Y. Kamay, and D. Laor, "KVM: The Kernel-Based Virtual Machine for Linux," in *Proc. of Ottawa Linux Symposium*, Ottawa, Canada, Jun. 2007.

[21] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proceedings of IEEE INFOCOM*, 2010.

[22] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast Transparent Migration for Virtual Machines," in *Proc. of USENIX Annual Technical Conf.*, Anaheim, CA, Apr. 2005.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," http://ilpubs.stanford.edu:8090/422/, 2001.

[24] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 199–212.

[25] M. Satyanarayanan, B. Gilbert, M. Toups, N. Tolia, D. O'Hallaron, A. Surie, A. Wolbach, J. Harkes, A. Perrig, D. Farber *et al.*, "Pervasive personal computing in an internet suspend/resume system," *IEEE Internet Computing*, pp. 16–25, 2007.

[26] B. Schmidt, "Supporting ubiquitous computing with stateless consoles and computation caches," Ph.D. dissertation, Stanford University, 2000.

[27] V. Shrivastava, P. Zerfos, K. won Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware Virtual Machine Migration in Data Centers," in *Proceedings of IEEE INFOCOM Mini-conference*, Shanghai, China, April 2011.

[28] A. Stage and T. Setzer, "Network-aware migration control and scheduling of differentiated virtual machine workloads," in *Proc. of ICSE Workshop on Software Engineering Challenges of Cloud Computing*, Vancouver, Canada, May 2009.

[29] F. Standard, "1037c. telecommunications: Glossary of telecommunication terms," *Institute for Telecommunications Sciences*, vol. 7, 1996.

[30] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA: ACM, 2007, pp. 331–340.

[31] R. Turetsky and D. Ellis, "Ground-truth transcriptions of real music from force-aligned midi syntheses," in *Proc. Int. Conf. on Music Info. Retrieval ISMIR*, vol. 3, 2003.

[32] H. N. Van, F. D. Tran, and J.-M. Menaud, "Autonomic virtual resource management for service hosting platforms," in *Proc. of ICSE Workshop on Software Engineering Challenges of Cloud Computing*, Vancouver, Canada, May 2009.

[33] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon, "Overdriver: Handling memory overload in an oversubscribed cloud," in *Proc. of ACM VEE*, Newport Beach, CA, Mar. 2011.

[34] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Predicting application resource requirements in virtual environment," 2008.

[35] T. Wood, P. Shenoy, and A. Venkataramani, "Black-box and gray-box strategies for virtual machine migration," in *Proc. of USENIX NSDI*, Cambridge, MA, Apr. 2007.

[36] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory buddies: Exploiting page sharing for smart colocation in virtualized data centers," in *Proc. of ACM VEE*, Washington, DC, Mar. 2009.