Automatic Generation of Workload Profiles Using Unsupervised Learning Pipelines

David Buchaca Prats, Josep Lluís Berral, *Member, IEEE*, and David Carrera, *Member, IEEE*

***Abstract*—The complexity of resource usage and power consumption on cloud-based applications makes the understanding of application behavior through expert examination difficult. The difficulty increases when applications are seen as "black boxes," where only external monitoring can be retrieved. Furthermore, given the different amount of scenarios and applications, automation is required. Here, we examine and model application behavior by finding behavior phases. We use conditional restricted Boltzmann machines (CRBMs) to model time-series containing resources traces measurements like CPU, memory, and IO. CRBMs can be used to map a given historic window of trace behavior into a single vector. This low dimensional and time-aware vector can be passed through clustering methods, from simplistic ones like *k*-means to more complex ones like those based on hidden Markov models. We use these methods to find phases of similar behavior in the workloads. Our experimental evaluation shows that the proposed method is able to identify different phases of resource consumption across different workloads. We show that the distinct phases contain specific resource patterns that distinguish them.**

***Index Terms*—Unsupervised learning, CRBM, deep learning, workload modeling, phase detection, MapReduce.**

I. Introduction

THE EXTREME complexity of current and future data centers, which are built from a large number of specialized technologies such as Non Volatile Memories (NVM), programmable circuits (FGPAs) and Graphical Processing Units (GPUs), poses a huge challenge: to develop technologies that allow for a holistic management of both workloads and the infrastructure while observing differentiated performance goals. The problem of mapping workloads on top of the hardware resources with the goal of maximizing both the performance of workloads and the utilization of resources, referred to as the placement problem, is well known for being NP-hard, with an underlying similarity to the multi-dimensional knapsack problem. The common approach used in the past has been to design heuristics that adapt to different contexts, providing vertical solutions for a given workload mix and underlying infrastructure, but which cannot be generalized. When the workload mix is completely heterogeneous, and the infrastructure hybrid and unexplored, the problem becomes even more challenging and needs to be automated.

In order to feed the heuristics used to manage data centers, it is a common practice to use workload models [1]–[3]. Application modeling is an active field in autonomic computing towards performance optimization. As computational resource sharing becomes critical, environment set-up and schedule must be tailored for each application. Unfortunately, applications are often provided as black-boxes, and modeling must be done through sampling executions in sandboxes [4], [5]. Furthermore, modeling must focus not only on single-running executions, but also on environments with several applications competing for shared resources. This implies that models not only need to characterize applications but also interference between them.

Existing literature in the area has studied the behavior of applications by attempting to understand common patterns across workloads, working on the assumption [3], [6], [7] that different but recurrent behaviors occur during the course of the execution, which is known as *phases*. Such phases display similar usage of computational resources over time. Recognizing which phases compose an application, and identifying the resource usages for each one, allows us to adapt the environment for a better performance as well as predict what applications can be co-located without interfering in their usage of resources. In this way, applications can be scheduled by means of decomposing them into phases instead of looking at their complete runtime.

While some works propose invasive techniques by placing *phase markers* in applications source code [8], [9], here we deal with black-box scenarios in which the application can only be monitored through resource consumption patterns. Workload activity is usually collected in the form of traces, which are usually logs for CPU, memory, disk or network usage, among others. Also, other traces related to the infrastructure can also be provided, such as energy consumption, utilization of GPUs or co-processors, and other custom metrics coming from the Operating System or external sensors and devices. They are therefore, in fact, multi-dimensional time-series.

Manuscript received May 15, 2017; revised September 15, 2017 and December 7, 2017; accepted December 10, 2017. Date of publication December 27, 2017; date of current version March 9, 2018. This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 639595). It is also partially supported by the Ministry of Economy of Spain under contract TIN2015-65316-P and Generalitat de Catalunya under contract 2014SGR1051, by the ICREA Academia program, and by the BSC-CNS Severo Ochoa program (SEV-2015-0493). The associate editor coordinating the review of this paper and approving it for publication was G. Casale. *(Corresponding author: David Carrera.)*

The authors are with the Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain, and also with the Data-Centric Computing, Barcelona Supercomputing Center, 08034 Barcelona, Spain (e-mail: david.buchaca@bsc.es; josep.berral@bsc.es; david.carrera@bsc.es).

Digital Object Identifier 10.1109/TNSM.2017.2786047

In this paper we present a novel approach to automatically finding and characterizing application behavior phases, using resource consumption traces as multidimensional time-series. In our method, we combine Conditional Restricted Boltzmann Machines (CRBM) [10], [11] and Hidden Markov Models (HMM) to distinguish changes on the resource consumption patterns over time. Our solution uses two models for two different goals. The CRBM is used to model the time-series and generate a code at every time step. This code summarizes information of the workload trace at time t as well as the information of the past n time steps. Then the HMM assigns a label to the code, automatically detecting and tagging different resource consumption patterns (see Section IV for further details).

Using this approach, workload traces can be mapped to a series of abstract phases that give a high level description of the resource consumption characteristics. Such a description can be used, for example, to identify interferences between workloads [12], [13]. The proposed technique is aimed at ingesting telemetry data (CPU, Memory, IO consumption, among others) to automatically characterize the behavior of workloads and dynamically produce workload profiles that can be leveraged by resource-aware resource schedulers (see Section II for more details). Moreover, the CRBM could be used as a generative model which would enable forecasting of both phase and resource consumption.

Nevertheless, forecasting is not the main focus of the work presented in this paper, the main contributions of which can be summarized as follows:

- Novel application of a combination of Conditional Restricted Boltzmann Machines (CRBM) and Hidden Markov Models (HMM) to encode time series and perform phase detection.
- Phase detection method based on unsupervised learning in time series (data center telemetry). The method is robust in front of burstiness in the time-series values (metrics), since phases are identified as HMM regimes that can be either stationary in terms of resource consumption or include periods of oscillation in a metric.

We believe that the combination of these two contributions is relevant for system management because phases characterize periods of a particular resource consumption pattern, and therefore are suitable for use by resource managers and workload schedulers to implement workload co-location strategies based on predictions.

The proposed method is evaluated using three different datasets, which are described in detail in Section V. The datasets comprise a mix of Big Data workloads involving Hadoop and Spark applications extracted from two well-established benchmarks: HiBench and TPCx-BB (BigBench). They represent numerous real-world applications, including MapReduce, Natural Language Processing, SQL and Machine Learning workloads, with different job lengths and data scale sizes. Additionally, and for sanity check purposes, the method is also tested against a well-known dataset containing human motion traces.

The experiments presented in this paper show that:

1) The combination of CRBMs and HMMs can be leveraged to automatically discover differentiated execution behaviors in the workloads. CRBMs provide the means to capture the time dimension of the input time series, reduce data dimensionality and expose the compressed data to the HMM. At the same time, the HMM extracts inter-phase patterns and automatically *tags* phases.
2) HMMs have a slight advantage over other well-known clustering techniques such as k-means in determining the phase from outputs of the CRBM, when comparing a-posteriori towards reference sources like changes in the Hadoop stages and resource consumptions.
3) Each discovered phase corresponds to a set of resource patterns.
4) Each different workload displays different phase patterns, to be exploited towards scheduling and workload characterization and identification.

The rest of the paper is structured as follows: Section III summarizes the state of the art and related work. Section IV presents the methodology, scenario and the techniques employed. Section V provides a description of the used data used for experimentation. Section VI shows the experiments performed to validate this work, and finally Section VII discusses the conclusions and future work.

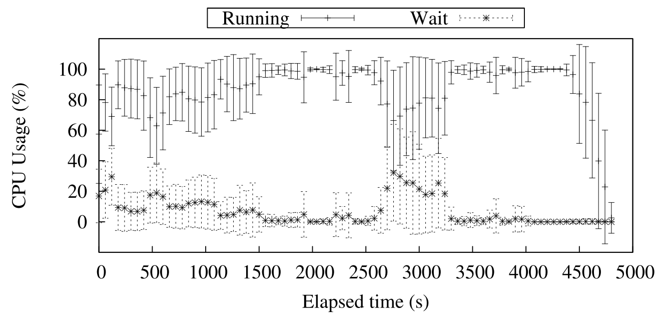
II. Motivation

Modern data centers keep growing in size on their way towards exa-scale clusters, which results in vast amounts of performance data (telemetry) being generated continuously. Microsoft [14] claims that their large data centers consist of more than 100,000 servers, each with a 10 to 40 Gbps network connection. At high utilization levels, their aggregate traffic can easily exceed 100 Tbps, and they perform analysis of this packet-level network telemetry to understand the traffic of their data centers. Netflix also reports that their Atlas [15] Telemetry platform was used to monitor 2 million metrics related to their streaming systems back in 2011, while in 2014 they reached 1.2 billion metrics, and these figures continue to rise as reported by them.

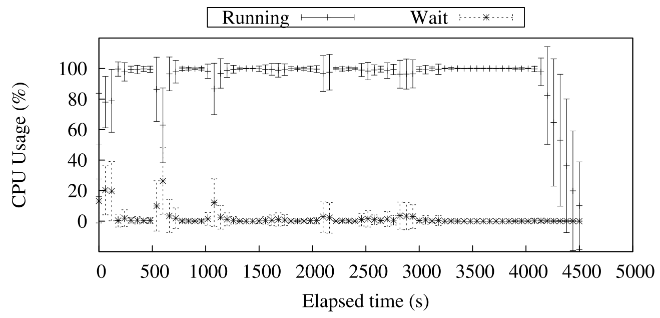
In order to manage such scenarios, workload scheduling mechanisms are leveraged to continuously optimize the existing deployments. Existing resource-aware job scheduling techniques [4], [5], [12], [13], [16]–[18] rely on the use of job profiles containing information about the resource consumption for each job.

Profiling is one technique that has been successfully used in the past for MapReduce [19] clusters. Its suitability in these clusters stems from the fact that, in most production environments, jobs are run periodically on data corresponding to different time windows [6]. Hence, profiles remains fairly stable across runs [7].

In this section we take the work presented in [17] as an example to illustrate the importance of accurate job profiles: the authors propose a novel Hadoop [20] scheduler that can allocate a variable number of tasks per node (TaskTracker in Hadoop terminology), as opposed to the usual approach (represented here by Hadoop Fair Scheduler) that provides for a static number of tasks per node. In the experiment, a combination of 8 different jobs are run using a standard FairScheduler and a Resource Aware Scheduler. Each job was previously



(a) Fair Scheduler



(b) Resource-aware Scheduler

Fig. 1. Example of Workload Co-location in terms of CPU utilization: (a) corresponds to Fair Scheduler using 4 slots per TaskTracker, with no resource-awareness; (b) corresponds to a Resource aware scheduler that determines a resource-aware strategy per TaskTracker that reduces resource contention.

profiled, comprising CPU, memory and network resource consumption over time. These profiles were made available to the Resource Aware Scheduler, as well as the capacities of the TaskTracker nodes. Under such circumstances, the Resource Aware Scheduler was able to compute the optimal number of tasks of each job that could be placed in each TaskTracker node, as well as the best task placement strategy, in order to make the most of the available resources. In practice, the scheduler could determine the best task co-location strategies and avoid resource over or under-commitment. The result of the experiment is that the Resource Aware Scheduler was able to complete the workload execution in a significantly shorter time.

To explain the reason for the improvement in performance, Figures 1(a) and 1(b) show the impact of performing a resource aware scheduling in terms of resource usage. These figures show the percentage of CPU time that TaskTrackers spent running tasks (either in system or user space), and the time that the CPU spent waiting. For each metric, we show the mean value for the cluster and the standard deviation across TaskTrackers. Wait time represents the time that the CPU remains idle because all threads in the system are either idle or waiting for I/O operations to be completed. Therefore, it is a measure of resource wastage, since the CPU remains inactive. While wait time is impossible to avoid entirely, it can be reduced by improving the overlapping of tasks that stress different resources in the TaskTracker. It is noticeable that in the case of the Fair Scheduler the CPU spends more time waiting for the completion of I/O operations than the resource-aware scheduler. The reasoning behind this result is that schedulers that are not resource-aware do not consider the resource consumption of applications when making task assignment decisions, and therefore they are not able to achieve good overlap between I/O and CPU activity.

In this paper we propose a technique for ingesting telemetry data (CPU, Memory, IO consumption, among others) to automatically characterize the behavior of workloads and dynamically produce workload profiles that can be leveraged by resource-aware resource schedulers. The proposed technique targets a highly dynamic environment, such as that described in [17], in which new jobs can be submitted at any time and in which workloads share physical resources among them. As will be shown through the experiments, we have verified that the method is suitable for workloads of different types, including MapReduce, Natural Language Processing, Machine Learning, and SQL queries. For this purpose we have leveraged two different datasets containing workload activity logs: one obtained from Hadoop jobs, and the other containing Spark jobs running TPCx-BB, which contains several types of applications. We used traces from on-premise clusters as well as virtualized Cloud-based workloads.

While the metrics used in this paper are limited in number (mainly CPU, memory, disk and network usage), there are many cases in which this list can be significantly longer. In several studies, such as [21], low level processor information is required to understand the root cause of performance degradation under different circumstances. Data provided by libraries like PAPI [22], widely used in the field of performance characterization and monitoring, can extract several hundred different counters from modern processors. Many of them are interrelated, such as L1/L2/L3 memory cache misses, but they are still relevant for many performance driven decisions. An example of such importance is the emerging effort to develop novel NUMA-aware [23] and GPU-topology-aware [24] placement strategies for BigData and DeepLearning workloads, because the topology of modern processors is extremely complex and can significantly impact the performance of applications. Furthermore works like [25] (which use the same features as we do in our experiments) use dimensionality reduction in order to facilitate parameter estimation. As the number of features grows, the amount of data needed to generalize accurately grows exponentially (this is known as the curse of the dimensionality). In these situations, and especially when the number of monitored features is large, techniques that provide dimensionality reduction like the one presented in this paper are relevant for building models that capture the different stages of execution of a given application.

III. Related Work

Workload modeling has been widely explored in the literature to produce more efficient resource management methods. Some existing works use simulations to generate models, such as in [1], but these approaches are limited in their applicability in real-world scenarios as they require complex simulations to generate workload patterns. Other works use a black-box approach based on the generation of workload profiles from previous executions, as in [2], where Esfandiarpoor *et al.* perform efficient workload collocation to save data center energy consumption. For this purpose they conduct VM analysis to determine the requested MIPS and memory of each VM arrived and VMs running on the system in each time interval. In some cases, user-provided phases can be introduced by the programmer, as in the case of [26], where Rosa *et al.* present a tool for workload modelling and reproduction parallel applications in which the user is responsible for building a task graph that is defined by a list of phases. Phases model different behaviours (CPU, IO, LOOP, FORK, JOIN). The main objective of this work is provide a tool for programmers to facilitate the understanding of parallel applications.

Other workload model construction techniques have been explored in the literature, like hierarchical sparse coding (a form of deep learning) to model user-driven workloads as presented in [3]. In that work, the authors use hardware performance counters to perform power scaling based on the workload characteristics. By using this technique they can differentiate different resource consumption phases on their tested workloads. While their work is similar to the one presented in this paper, they only focus on processor counters, while we go beyond that method by including IO and memory consumption metrics at a cluster level.

A relevant aspect related to workload modeling and scheduling is the ability to predict workload interference in order to define co-location and anti co-location strategies. This topic has been studied in the literature, and several works address it using different approaches. Mishra *et al.* [12] propose a novel machine learning method to predict application interference. They collect data from the performance counters of the processors to model interference and leverage the models to improve scheduling efficiency. Nevertheless, they use synthetic application kernels for the evaluation, ignoring the different phases of resource consumption that can be observed in real-world workloads, whereas in our work we use real applications, not just kernels. A similar approach is used in [13], where a set of benchmarks were used to quantify resource interference across co-located workloads. Using a scheduling method, the authors identify interferences and leverage that information to improve workload management. Our work is complementary to that one in the sense that adding precision to the workload characterization, identified by phases in our work, would allow for more fine grained scheduling decisions. Some existing works rely on predictive behavior on resource sharing environments to enforce Quality of Service (QoS) guarantees, as in the case of [18]. Finally, other approaches on sandbox experimentation perform real experiments in isolated environments using real workloads to find optimal resource allocations, as in [4] and [5].

The ability to make look-ahead predictions on expected phase changes over time is an important control knob that can be leveraged for more accurate resource management as shown in [16] and already discussed in Section II. Phase detection has been extensively studied, using both supervised and un- supervised techniques towards finding behavior changes in workloads.

Ding *et al.* [27] focus on applications phase detection and exploitation by means of two approaches, top-down and bottom-up, also taking into account off-line and on-line phase detection. In the top-down approach, execution is divided into candidate phases, based on the high-level structure of the source code. The beginnings of long-running subroutines and loops mark the potential boundaries between phases. Such an approach requires compile-time instrumentation to insert marks at candidate phase boundaries. The bottom-up approach starts with the behavior metrics observed during execution and looks for recurring patterns and changes. The beginning of long-running subroutines and loops marks the potential boundaries between phases. This can be done with unmodified program binaries, yet is likely to be strengthened considerably by going back to the source code to correlate observed phase transitions with certain groups of static instructions. However, profile-driven strategies like the ones explained in such works require the insertion of markers into the code, and this implies being able to access it. In our current approach, we focus on total non-invasion and preemptive knowledge, since the running application is presented as a black-box, where our data comes from the profile of the resources accessed by it.

Pipada *et al.* [28] present a method for learning to identify workload phases from live traces using Support Vector Machines (SVMs) to classify phases that have been manually tagged from a Dataset of Storage traces. The ultimate goal of the paper is to trigger phase-specific system tuning for disk IO time series. The main drawback is that data must be manually tagged, so in the scenario presented there the learning process would require supervision from the application owner. The method is evaluated using accuracy across all classes.

Hidden Markov Models are also used for phase analyses on executions in works, as in [8], which uses HMMs to model phase behavior via branch-instruction traces generated during the program executions. The authors pre-process the branch-instruction traces by binning together discrete observations from windows into a vector. The vector for a given window contains the number of times at each component that a particular symbol appeared. This process maps the windows discrete observations into a single vector. The main drawback is that the original ordering of symbols is lost at granularities smaller than the window size. Since this is unsupervised learning, the data does not contain tags for phases, as the user specifies how many phases are expected to be found as hidden states on the HMM. As in our work, they train the HMMs using the Baum-Welch algorithm (which is the EM algorithm applied to HMMs) [29]. The evaluation is conducted by measuring how much variance can be accounted for by the language and state probability transition matrices, then computing the accuracy with respect to their "prior-model“. In contrast with our HMMs, their data consists of symbol time-series turned into real values to feed modified HMMs (CD-HMM and the VQ-HMM), while we use the CRBM representation of inputs to feed our HMM.

Finally, Nagpurkar *et al.* [9] focus on on-line phase detection algorithms. Their work also uses the source code to identify loops and repeated method invocations to build a baseline solution. It then compares the proposed phase modeling against the baseline solution. In order to identify periods of repetition (and then phases), loops and method invocations are selected from the source code and the entrance and exit of each repetition construct is recorded with a unique identifier. Their unsupervised learning methodology uses the minimum phase length as hyper-parameters, rather than determining the number of expected phases to be found. They also require the source code for such analyses.

As here explained, many methods attempting phase detection on application executions employ source code analysis or marking. This involves several drawbacks: such a process is tedious and specific to each source code; moreover, in provided data-center and "cloud" scenarios the source code is not available, since applications are submitted as black-boxes. Our approach focuses on attacking the problem from resource usage logs and sensors, totally non-invasive towards the application and available from the cloud provider point of view. Also, instead of feeding HMMs with direct telemetry data, we pre-process it using CRBMs, which have already been used for modeling complex multidimensional sequential data such as human motion data [10] or financial data [30]. To the best of our knowledge, no other work has used CRBMs in combination with clustering methods for phase detection.

IV. Methodology

Defining phases for time series is not a trivial task. Workload traces contain complex non-linear relationships between the different components of CPU, RAM, Memory and Disk, so defining phases of similar behavior over time becomes a very challenging task. In order to facilitate this, we learn a representation that maps slices of those multidimensional sequences into vectors. This section describes how this is done using a CRBM. Then we use a Hidden Markov Model trained on the learned features to find meaningful phases in this new representation. Finally we compare the predicted phases with the meta-information we have obtained from workload indicators to verify the results.

In scenarios like the one proposed, where no true labels exist, or existing labels are either approximate, inaccurate or too generalized, evaluating phase detection models is not trivial. For example, Hadoop executions have labeled "stages" indicating the predominant type of task being executed at each moment ("map", "reduce", "shuffle", ...). In this example, throughout their execution Hadoop workloads present different behaviors along their execution that change depending on the stage and on the application itself. In other words, two different workloads will present different behaviors for the same stage, but two similar ones will behave similarly.

Nevertheless, we can evaluate the quality of the phase prediction on workloads by computing the accuracy between predicted phases against labels on meta-data execution. Such metrics will not be indicative of the phases to be discovered, since this learning method is unsupervised for discovering unlabeled behaviors. However, they will indicate how plausible it is to use the produced phases to gauge the little information the application is providing about their execution stage. We would like to recall that the goal of this work is to learn phases in situations where labels may not be available. So, in this case the Hadoop meta-data is used only for side-validation but never as a target feature for supervised learning.

To assess the quality of the phases proposed by the method, we check the correspondence between detected phases and different resource usage. In Section VI, comparisons will be made to show how similar and different types of workload with different execution stages each are detected and identified.

A. Representation Vectors

Let us consider a set of M sequences $\boldsymbol{X}$. In our application each sequence $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_l) \in \boldsymbol{X}$ will contain measurements from the execution of a program. The length of $\boldsymbol{x}$, is equivalent to the execution time (in seconds) of the workload. Each component $\boldsymbol{x}_t$ is a vector in $\mathbb{R}^{n_v}$, where n_v is the number of features (or measurements) used to describe the sequence at each time step. Notice that sequences are not required to have the same length.

Instead of using directly the sequences from $\boldsymbol{X}$, or manually defining features that aggregate resource consumption over time, we propose to learn a vector representation for our sequence components. A vector representation is a function $\phi : \mathbb{R}^{n_v} \longrightarrow \mathbb{R}^{n_h}$ that maps the original measurements of $\boldsymbol{x}$ at time t, $\boldsymbol{x}_t$, to a vector of length n_h. Given a sequence $\boldsymbol{x} \in \boldsymbol{X}$ we will map $\boldsymbol{x}_t \in \mathbb{R}^{n_v}$ to $\phi(\boldsymbol{x}_t; \boldsymbol{\theta}) \in \mathbb{R}^{n_h}$. The parameters $\boldsymbol{\theta}$ of the representation will be learned from the data, with the goal of maximizing the probability of the sequences in $\boldsymbol{X}$. The n_h value is a hyper-parameter of the representation.

Since our data is composed of sequences we would like the mapping $\phi(\boldsymbol{x}_t; \boldsymbol{\theta})$ to depend on $\boldsymbol{\theta}$ but also on the previous n components of the sequence. This means $\phi\big(\boldsymbol{x}_t; \boldsymbol{\theta}\big) = \phi(\boldsymbol{x}_t; \boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-2}, \ldots, \boldsymbol{x}_{t-n}, \boldsymbol{\theta})$. Notice that for the first n values we cannot use this mapping since there are not enough measurements for ϕ. One way to fix this problem would be to set the first history values to zero. Another option is to simply not to use ϕ for the first n time steps. This paper explores the use of a CRBM as a good candidate for $\phi(\boldsymbol{x}_t; \boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-2}, \ldots, \boldsymbol{x}_{t-n}, \boldsymbol{\theta})$.

B. Restricted Boltzmann Machine

A CRBM is an extension of a Restricted Boltzmann Machine (RBM) especially designed to handle sequential data. Before dealing with the time dependence, we will present how to model static frames of the time-series data. The CRBM presented in Section IV-C uses a Gaussian Bernoulli RBM (GB-RBM) to model the static frames $\boldsymbol{x}_t$ of the time series. This work does not use a standard RBM, because the workload data used in the experiments is made of real valued components and the standard RBM models binary valued data.

The GB-RBM is an Energy-Based Model with Gaussian visible variables $\boldsymbol{v}$ and hidden Bernoulli variables $\boldsymbol{h}$. Variables in this type of models are also called "units" or "neurons". We used the same GB-RBM as in [10] and [31]. The joint log-probability $p(\boldsymbol{v}, \boldsymbol{h})$ defined by the model is given by the

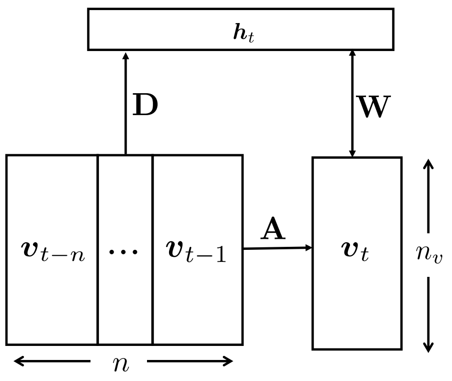


Fig. 2. CRBM diagram.

following expression:

$$\log P(\boldsymbol{v},\boldsymbol{h}) = \sum_{i=1}^{n_v} \frac{(v_i-c_i)^2}{2\sigma_i^2} - \sum_{j=1}^{n_h} b_j h_j - \sum_{i=1}^{n_v}\sum_{j=1}^{n_h} \frac{v_i}{\sigma_i} h_j w_{ij} + C \qquad (1)$$

where σ_i is the standard deviation of the Gaussian for unit v_i, w_{ij} is the weight connecting visible unit i with hidden unit j, $\boldsymbol{c}$ is the bias of the visible units, $\boldsymbol{b}$ is the bias of the hidden units and C is a normalization constant. We fixed, σ_i to 1 (for all i), based on the works of Taylor *et al.* [10].

The model parameters are learned using mini-batch stochastic gradient descent. The gradient of the log-likelihood of the data can be approximated using the contrastive divergence (CD-K) algorithm [32]. In our experiments we used 1 step of Gibbs sampling to generate the negative phase (we used CD-1 with momentum).

C. Conditional Restricted Boltzmann Machines

The CRBM is essentially a GB-RBM with some extra connections used to model temporal dependencies. To model such dependencies, the CRBM keeps track of the n previous visible vectors, which are kept in $\mathbf{His}^n$. We will call $\mathbf{His}^n$ the *history* of the CRBM.

The parameters of the CRBM are $\boldsymbol{\theta} = \{\boldsymbol{W},\boldsymbol{A},\boldsymbol{D},\boldsymbol{c},\boldsymbol{b}\}$. $\mathbf{W},\mathbf{A},\mathbf{D}$ are matrices and $\mathbf{c}$, $\mathbf{b}$ are the vectors of biases for the visible and hidden units, respectively. $\mathbf{W} \in \mathbb{R}^{n_h \times n_v}$ models the connections between visible and hidden units. $\mathbf{A} \in \mathbb{R}^{n_v \times (n_v \cdot n)}$ is the mapping from the history to the visible units. $\mathbf{D} \in \mathbb{R}^{n_h \times (n_v \cdot n)}$ is the mapping from the history to the hidden units.

Let us consider a multidimensional sequence $\boldsymbol{v} = (\boldsymbol{v}_1,\boldsymbol{v}_2,\boldsymbol{v}_3,\ldots,)$. The history for $\boldsymbol{v}$ at time t, denoted by $\mathbf{His}_t^n$, is defined as $(\boldsymbol{v}_{t-n},\ldots,\boldsymbol{v}_{t-1})$ and contains the previous n vectors from time $t-1$ to $t-n$. At time $t+1$, vector $\boldsymbol{v}_t$ is pushed into the history while observation $\boldsymbol{v}_{t-n}$ is popped out. Therefore $\mathbf{His}_{t+1}^n$ is $(\boldsymbol{v}_{t-n-1},\ldots,\boldsymbol{v}_t)$. Notice again that such a mechanism needs the first n observations of each time series to have enough data to properly fill the *history* structure. Figure 2 shows a diagram of the CRBM.

TABLE I
CRBM GRADIENT OF THE PARAMETERS

Parameter	Gradient Approximation
$\boldsymbol{W}$	$\boldsymbol{h}\cdot\boldsymbol{v}^T - \hat{\boldsymbol{h}}_{(k)}\cdot\hat{\boldsymbol{v}}_{(k)}^T$
$\boldsymbol{A}$	$\boldsymbol{v}\cdot\mathbf{His}^T - \hat{\boldsymbol{v}}_{(k)}\cdot\mathbf{His}^T$
$\boldsymbol{D}$	$\boldsymbol{h}\cdot\mathbf{His}^T - \hat{\boldsymbol{h}}_{(k)}\cdot\mathbf{His}^T$
$\boldsymbol{c}$	$\boldsymbol{v} - \hat{\boldsymbol{v}}_{(k)}$
$\boldsymbol{b}$	$\boldsymbol{h} - \hat{\boldsymbol{h}}_{(k)}$

Given a vector $\boldsymbol{v}$, we can obtain the hidden activation $\boldsymbol{h}$ by computing the sigmoid of incoming signal from $\boldsymbol{v}$ and $\mathbf{His}^n$, weighted by $\boldsymbol{W}$ and $\boldsymbol{D}$, respectively, and adding the bias of hidden units:

$$\boldsymbol{h}_{(n_h,1)} = \sigma\Big(\boldsymbol{W}_{(n_h,n_v)}\cdot\boldsymbol{v}_{(n_v,1)} + \boldsymbol{D}_{(n_h,n_v\cdot n)}\cdot\mathbf{His}^n_{(n_v\cdot n,1)} + \boldsymbol{b}_{(n_h,1)}\Big)$$

the subscripts in the previous line (written only for clarity reasons) indicate the dimensions of the different matrices and vectors. For brevity, we will express this without subscripts as

$$\boldsymbol{h} = \sigma\big(\boldsymbol{W}\cdot\boldsymbol{v} + \boldsymbol{D}\cdot\mathbf{His}^n + \boldsymbol{b}\big)$$

Notice that $\mathbf{D}$ defines a function from $\mathbb{R}^{n_v\cdot n}$ to $\mathbb{R}^{n_h}$ and $\mathbf{His}^n$ is expressed as a column vector of length $n_v \cdot n$ instead of a matrix of shape (n_v, n).

Inference in a CRBM is quite similar to an RBM. We can use stochastic gradient ascent to find parameters that yield models with high log-likelihood. Contrastive Divergence [32] can be used to find approximate gradients of the loss with respect to the parameters. Using CD-k, we end up with the approximate gradients shown in Table I. In this table, vectors $\hat{\boldsymbol{v}}^{(k)}$ and $\hat{\boldsymbol{h}}^{(k)}$ are values after k steps of Gibbs sampling of a vector $\boldsymbol{v}$ and $\boldsymbol{h}$, respectively. All quantities in the Gradient Approximation are taken at the same time t, which is omitted for brevity. More details about the fundamentals of CRBMs can be found in Taylor's work [10].

D. Data Pipeline and Architecture

Once the CRBM is trained we can compute the vector representation $\phi(\boldsymbol{x}_t;\boldsymbol{x}_{t-1},\boldsymbol{x}_{t-2},\ldots,\boldsymbol{x}_{t-n_{\text{his}}},\boldsymbol{\theta})$, or simply $\phi(\boldsymbol{x}_t;\boldsymbol{\theta})$, at every time step t using (2).

$$\phi(\boldsymbol{x}_t;\boldsymbol{\theta}) = \sigma\big(\boldsymbol{W}\cdot\boldsymbol{x}_t + \boldsymbol{D}\cdot\mathbf{His}_t^n + \boldsymbol{b}\big) \qquad (2)$$

Then we can discretize the result to get a binary code. The binary code is used as input to an HMM. We have chosen HMMs, instead of simpler unsupervised algorithm such as k-Means, because the HMM captures dependencies across time and therefore is suitable for our sequential data.

Given a number of hidden states, which correspond to the number of distinct phases we expect to exist, the parameters of the HMM are found using the Baum-Welch algorithm [29]. Once the parameters are learned, the most likely state sequence for a given observation sequence is efficiently found using the Viterbi algorithm [33].

After having trained both the CRBM and the HMM, the pipeline for a given sequence $\boldsymbol{x}^m$ of length l_m is composed of the following steps.

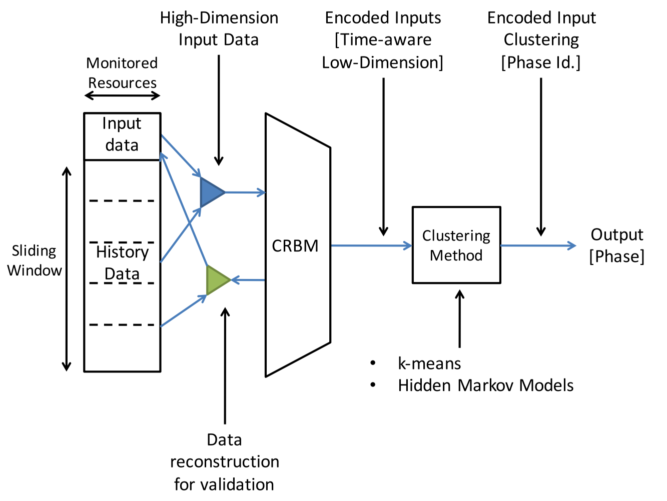

High-Dimension Input Data
Encoded Inputs [Time-aware Low-Dimension]
Encoded Input Clustering [Phase Id.]
Monitored Resources
Input data
Sliding Window
History Data
CRBM
Clustering Method
Output [Phase]
• k-means
• Hidden Markov Models
Data reconstruction for validation

Fig. 3. Data pipeline schema showing how the resource monitoring data (which contains the input and history data) passes through the CRBM and the clustering method.

TABLE II
EXAMPLE OF DATA SLICE FROM THE ALOJA DATASET. FOR NON-AVAILABLE VALUES, −1 IS USED INSTEAD

instant	pc.user	kbmemused	rxpck.s	tps
9	11.370	18,730,504	333	−1
10	3.110	18,782,464	276	−1
11	0.930	18,791,856	332	−1

- Step one: the representation $(\phi(\boldsymbol{x}_n^m; \boldsymbol{\theta}), \ldots, \phi(\boldsymbol{x}_{l_m}^m; \boldsymbol{\theta}))$ for the sequence $\boldsymbol{x}^m$ is computed using (2).
- Step two: the Viterbi algorithm is applied to the previous sequence to get the most likely state sequence $(y_n, \ldots, y_{l_m})$.

This approach does not give phase assignment to the first n components of the sequence, which we will consider as the "initial phase".

Figure 3 shows a diagram of the data pipeline. The input data and the history data are fed to the CRBM (at every time step). Then the CRBM gives a code the clustering method that outputs a phase.

V. DATASETS

A. Workload Dataset Description

The workloads used in the following experiments belong to the ALOJA Project[1] [34], [35], a repository of Big Data executions focused on benchmarking different infrastructure as well as software components. For each registered execution, the dataset contains the obtained monitors from CPU, memory, network and disks, among other execution details, e.g., markers for Hadoop, Spark, Hive, etc, with more than a record per second during the execution.

Table II shows a slice of 3 time steps from a workload extracted from the data. Data is aggregated per second, averaging the data when numeric. Column "instant" is used to identify the time in the series, but it is not used as an input for the machine learning pipeline. The selected features for this approach are "pc.user", "kbmemused", "rxpck.s" and "tps", corresponding to user process CPU usage (in % usage), Memory usage (in kilobytes), Network usage (received packages per second), and Disk usage (transactions per second).

It is true that other features can be added such as *transmitted* packages per second, or system process CPU usage, as well as maximum and minimum values for each feature, in addition to these. However, for this first proof of concept we decided to keep the input simplistic by selecting the most representative measurements from the workloads. The use of an extended feature version of this approach is intended for future work.

To simplify the feature naming, we will refer to the features as CPU, Memory, Net and Disk. As the workload is distributed among machines and processors, the CPU % usage is a sum over all used cores, and therefore can take values above 100.

B. Dataset A: Hadoop Workloads Using BigBench

The first dataset, extracted from ALOJA Hadoop Time-Series Dataset v1, contains 182 series from Hadoop executions (up to 22 different features at this time), from the Intel HiBench [36] benchmark suite. These workloads contain Map-Reduce algorithms for sorting (*Sort* and *Terasort*), word counting (*wordcount*), machine learning (*k-means* and *bayes*), input-output stress tests (*dfsioe-read*, *dfsioe-write*), etc. All the jobs have been running in on-premise infrastructures, with similar Hadoop configurations. Data generation jobs, usually accompanying workloads, have been excluded from the experiments.

C. Dataset B: Spark Workloads Using TCPx-BB

The second dataset, extracted from ALOJA Spark Time-Series Dataset v1, comprises 900 executions of 30 different Spark applications contained in the TPCx-BB (BigBench [37]) benchmark. TPCx-BB contains 30 frequently performed analytical jobs in the context of retailers with physical and online store presence. They represent different types of workloads (including Natural Language Processing, SQL queries, Mapreduce jobs and Machine Learning workloads), comprise different data types (Structured, Semi-Structured and Unstructured data), provide a mix of long and short running jobs and can run at different data scales (in our case, 1, 10 and 100GB). For each of the queries, we included 30 instances, comprising the different data scales mentioned before. All the jobs were run in the Microsoft's Azure cloud using Spark 2 as the engine. We used HDInsight PaaS to spawn the spark clusters, running a 16-slave node cluster (plus several redundant head nodes). Data was stored in the Azure Data Lake Store of Azure.

D. Dataset C: Human Motion Dataset

For sanity check purposes, in the final experiment presented in this paper we leveraged the well-established *Motion* dataset from Hsu *et al.* [38], also used in Taylor's CRBMs validation [11], to validate our method against a well-known dataset.

[1]Obtained from ALOJA Time-Series http://bscdc-login.bsc.es/alojaml.

TABLE III
CRBM Training Time, n_v Is the Number of Hidden Units. All Models Have the Same Delay, 50 Time Steps

n_h	3	10	50	100	200	300
minutes	11.3	13.0	23.3	36.3	99.9	136.7

VI. Experiments

Here we introduce six experiments for validation and testing of the presented approach. The following experiments describe how the proposed methodology differentiates phases throughout workload executions on different scenarios or types of tested workloads. Notice that, for the following figures, two kinds of plots are produced: detected phases and resources usage. For the detected phase plots *barplots* are used, where each phase is differentiated by color and also height, not as a significant value but as a visual aid for differentiating classification over time. As for the obtained phases, we will refer to as *phases* the tags given by the k-means algorithm and the tags given by the HMM as regimes, representing what we use from HMMs as phases.

In terms of datasets (see Section V), Experiments 1-4 use dataset A (Hadoop workloads), and Experiment 5 uses dataset B (Spark workloads). Experiment 6 uses dataset C (human motion identification) as a sanity check of the proposed method based on classical literature in the field.

A. CRBM Train Time

Table III shows the time needed to train a CRBM on the presented dataset, having randomly selected 66% series for training and 33% for testing, using a history length of 50 samples and different configurations of hidden units. All the training times presented measure 300 epochs of stochastic gradient descent with momentum of 0.4 and learning rate 0.001. For benchmark purposes, no early stopping is applied and the presented times use a single thread of CPU. Therefore, since most of the time is consumed by Matrix multiplications, train time can be speeded up approximately by a factor of the number of threads. The train time could be reduced by computing all matrix operations using GPUs.

We have found that the reconstruction error plateaus during the first 40 epochs and further training does not help. The reconstruction error achieved by the model using 100 hidden units is not significantly improved by models with more hidden units and the same history size.

B. Exp1: Unsupervised Automatic Phase Detection

In order to understand the different behaviors found in the predicted clusters given by the k-means and the HMMs, here we show some workloads with the associated tag sequences (the discovered phases). Although we have generated the study for all the workloads available in the data-set, we display here the most representative ones. After exhaustive experimentation with the CRBMs, we selected $n = 50$ as required "history length" to start encoding and predicting. The history period is marked in Figures 4, 6, and 7 by a vertical red line in the workload trace that marks the time $n = 50$.

Moreover, for the following experiments, several values of k (for the *k-means*) and expected regimes (in the HMMs) have been tested. The most distinctive value found for this hyper-parameter is 5 (clusters), since for lower values of k the algorithm displayed randomly-joined phases, while for higher values it converged by returning empty or underpopulated clusters. This led us to choose $k = 5$ as the fittest value for the current kind of workloads. Notice that for other kinds, this hyper-parameter must be tuned.

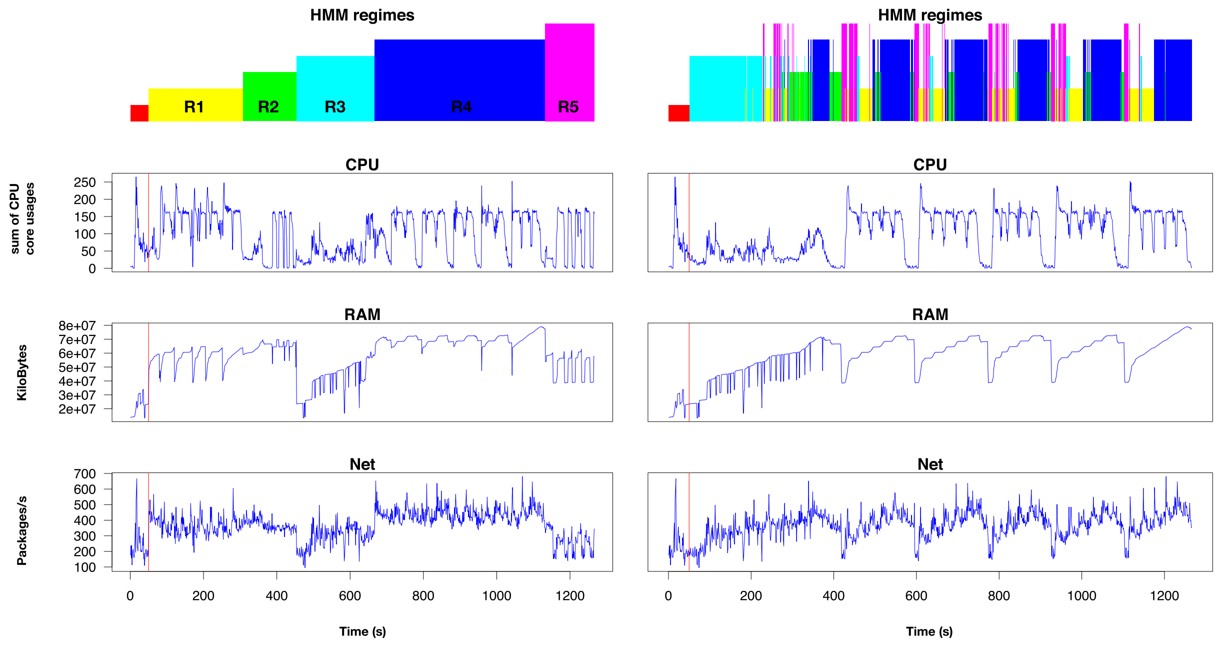
Figure 4 shows a couple of workload traces with the predicted phases $R1, \ldots, R5$ given by the HMM. The right hand side images from Figure 4(a) and Figure 4(b) contain the workload resource usage and the predicted phases in chronological order. The left hand side images contain the same information of CPU, Memory and Net traces, but grouped by the phase tag in order to see how each resource behaves in each given phase.

The aim of grouping the time-series elements by phase is to display the general trend of consumption for each resource, defining the phase. We have the supported hypothesis that each discovered phase will be characterized by a trend in one or more resources distinguishable from the other phases. The fact that usage in some resources does not need to be constant is covered by the encoding done through the CRBM. The left hand side images provided in Figure 4 are precisely created to visually aid distinction among different behaviors in the time-series. For a detailed visual inspection, Figure 5 contains the histograms of the different traces across all data, grouped by $R1, \ldots, R5$. Table IV contains the mean and standard deviation of the different trace components grouped by $R1, \ldots, R5$. The following brief description is a simplified textual description of those behaviors.

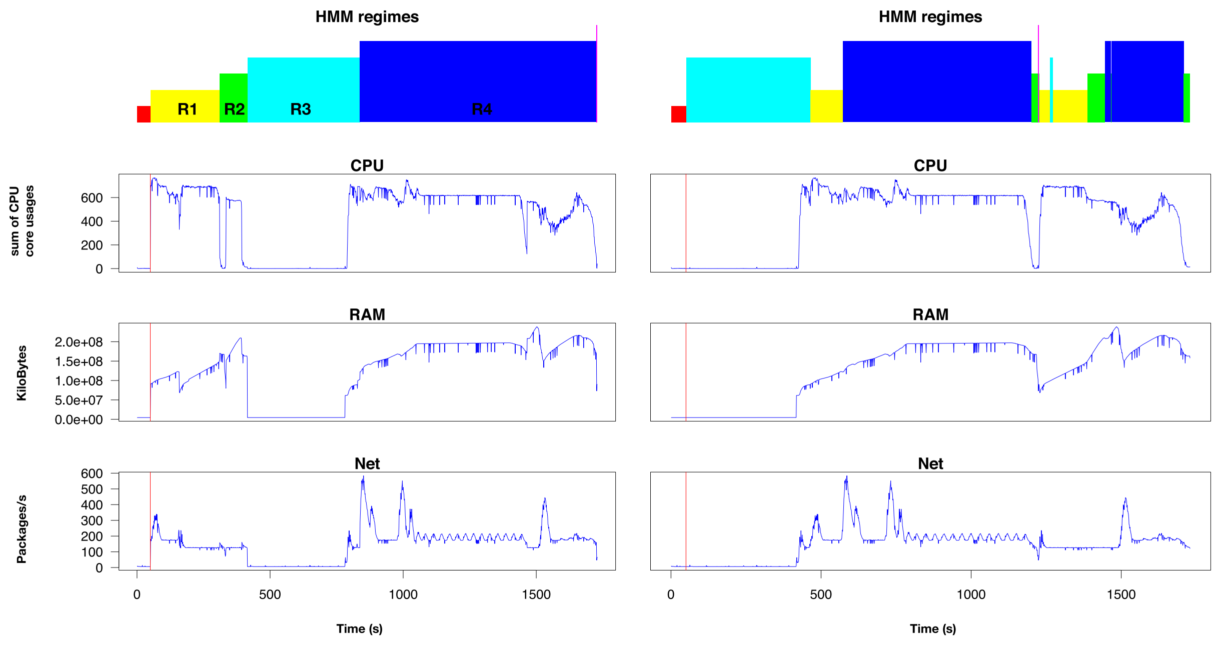
- $R1$ contains trace behavior with high CPU usage and high variance across all other traces. This pattern may be observed on the left-hand side workload in Figure 6, which shows the model detecting phase $R1$ around time step 1300, where there is a peak of CPU usage. Table IV shows that $R1$ has the highest mean CPU usage.
- $R2$ is similar to $R1$, but the Memory usage under R2 is higher and the CPU usage is slower. Table IV shows that $R2$ contains the second highest mean Memory usage.
- $R3$ detects regions with low Memory usage with low CPU usage. Table IV shows that $R3$ contains the lowest mean Memory usage and the second lowest Net usage.
- $R4$ contains high Memory, high Net usage. Table IV shows $R4$ as containing the highest mean Memory, Net and Disk usage.
- $R5$ contains similar behavior to $R2$ but with lower resource usage than $R2$.

C. Exp2: Phase Detection From Workload Traces

It is important to notice that Hadoop stages do not determine the behavior of the CPU, Memory and Net traces. Figure 6 shows two workloads with different Map, Reduce and Shuffle stages, containing similar behaviors in the traces for different stages. The vertical boxes in the figure show a slice of "$R4$" behavior with high Memory and above average Net usage



(a) Workload A. Left: Time series grouped by HMM Regime; Right: original time series



(b) Workload B. Left: Time series grouped by HMM Regime; Right: original time series

Fig. 4. Both Workload A and Workload B contain, on the right hand side, the true traces and predicted regimes. On the left hand side are the traces clustered by tag to facilitate visual inspection of similar trace behaviors. Red regions/lines mark the "delay" data required to start the encoding.

taking place in two different Hadoop stages (*map* for workload 1 on the left, and *reduce* for workload 2 on the right).

The presented methodology is not intended to detect Hadoop stages as "phases", but for the same kinds of workloads it detects the same phases for the same stages, while for different workloads, for the same kind of behavior it detects the Hadoop stages that behave similarly to one another. This allows us to characterize applications according to sequences of phases during the execution. As the methodology presented herein never sees the Hadoop stages, it relies on the provided resource traces, which makes it extensible to any other application and framework.

D. Exp3: Accuracy Analysis Using Hadoop Logs: Mapping Detecting Phases to MapReduce Phases

The Map-Reduce data used contains several tags at each time-step. Tags have been used only for evaluation purposes (not for training the algorithms). We have previously remarked that Hadoop phases do not determine the resource consumption, as can be seen in Figure 6. Nevertheless, we can make

TABLE IV
MEAN AND STANDARD DEVIATION OF THE NORMALIZED TRACES UNDER THE DIFFERENT REGIMES GIVEN BY THE HMM. VALUES IN BOLD ARE THE HIGHEST ACROSS THE COLUMNS

Regime	CPU.mean	Mem.mean	Net.mean	Disk.mean	CPU.std	Mem.std	Net.std	Disk.std
$R1$	**0.724**	−0.379	0.519	−0.015	**1.515**	**0.866**	**1.062**	**0.399**
$R2$	−0.348	0.539	−0.222	0.032	0.766	0.554	0.526	0.303
$R3$	−0.186	−0.857	−0.710	−0.060	0.732	0.834	0.588	0.342
$R4$	0.341	**0.585**	**0.946**	**0.047**	0.698	0.726	0.753	0.270
$R5$	−0.527	0.045	−0.731	−0.020	0.677	0.757	0.349	0.320

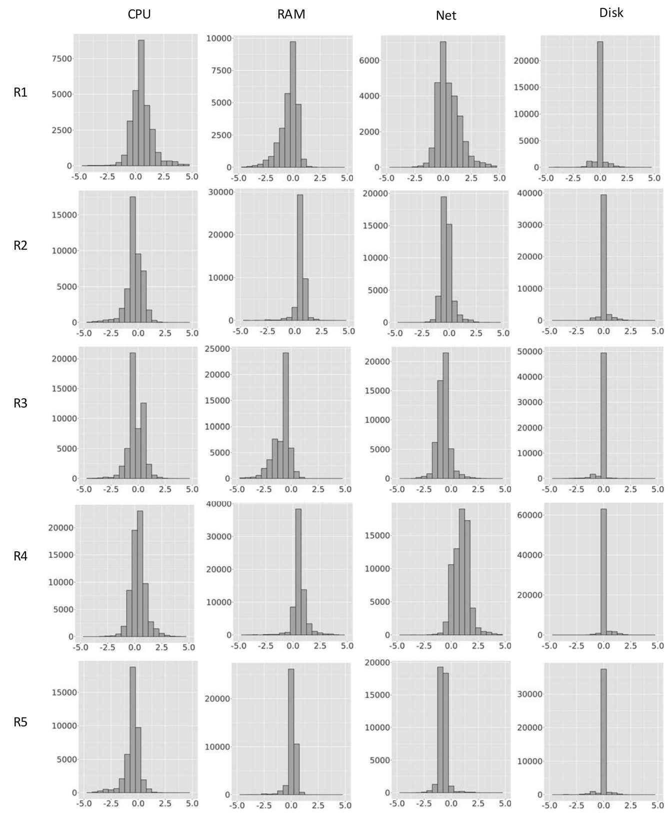


Fig. 5. Histograms for CPU, Memory, Net and Disk normalized traces under each of the automatically generated phases (or regimes) $R1, \ldots, R5$. The value 0 on the x axis can be interpreted as the mean value of a feature.

an approximate validation of our model by comparing the predicted phases with the Hadoop phases. Moreover, we can compare the phases given by the k-means and the HMM in the learned representation.

The most representative phases of this type of workloads are the map phase, the reduce phase and the shuffle phase. We have codified the tags as integer values, which we will refer to as the true tags. The codification of the true tags has been performed as follows. Let us consider binary valued vectors (m, s, r) where each index taking value 1 represents that the data is in a particular state. The use of this form $(1, 0, 0)$ represents that the data is in a map state, $(0, 1, 0)$ in a shuffle state and $(0, 0, 1)$ in a reduce state. Any other combination represents data in a combination of states; for example, $(1, 1, 0)$ would represent the data being in a map and shuffle phase. Each possible binary vector has been assigned to an integer, the equivalent number in binary form. For example, $(0, 0, 1) = 1$ and $(1, 0, 1) = 5$.

TABLE V
ACCURACY RESULTS OF THE BEST ALIGNMENT BETWEEN TRUE AND PREDICTED PHASES

k clusters	k-means train	hmm train	k-means test	hmm test
2	0.447	0.449	0.498	0.494
3	0.491	0.493	0.507	0.537
4	0.490	0.511	0.464	0.547
5	0.506	0.531	0.483	0.547
6	0.344	0.461	0.302	0.472
7	0.409	0.440	0.447	0.452

E. Exp4: Finding a Correspondence Between True Phases and Predicted Phases

To assess numerically the quality of our phases, we find for each value of k (number of clusters) the correspondence that most closely matches the predicted phases and the true phases. That is, we find a matching function f^* that maximizes the accuracy of the predicted phases and the true phases across all our data. Let $\boldsymbol{Y}$ be the set of sequences containing the correct phases. Let l_y indicate the length of a sequence of phase tags $\boldsymbol{y} \in \boldsymbol{Y}$. Then, the best matching between the predicted and the true phases is

$$f^* := \arg\max_f \sum_{\boldsymbol{y}\in\boldsymbol{Y}} \sum_{j=1}^{l_y} \mathbb{1}_{(y_j = f(\hat{y}_j))} \tag{3}$$

where f is an injective function from the first k integers to the total number of true distinct phases. Notice that f has to be injective, since we do not want to allow naive solutions where two distinct predicted clusters are aligned to the same "true cluster". Results of the best alignments for $k \in \{2, \ldots, 7\}$ can be found in Table V.

Both k-means and HMM models achieve similar results, but the HMM obtains consistently better accuracy in both training and test sets across all number of clusters, which shows that according to this intrinsic evaluation it is a better model for this type of data. This result is consistent with our prior knowledge about the model. The HMM hidden states take into account the previous hidden states when generating a phase sequence. The k-means is not aware of any time-dependencies when proposing phases, although the representation that is fed to the k-means summarizes historical information.

F. Exp5: Validity of Model Across Workloads

Here we present the application of the method for phase detection results on more heterogeneous non-Hadoop set of workloads (Spark dataset), to demonstrate that the presented approach can be applied of different kinds of jobs, such

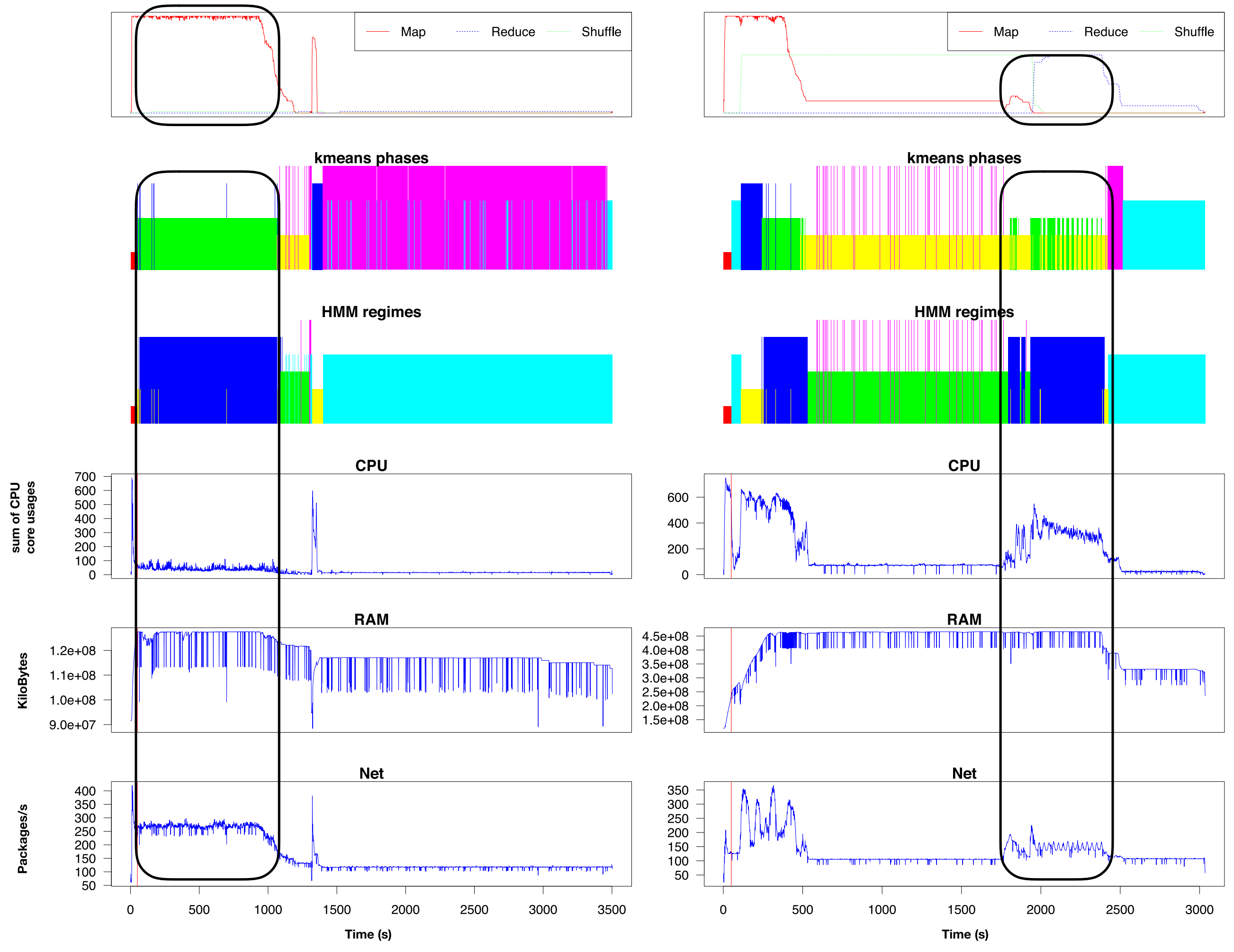


Fig. 6. Two different workloads side by side. The vertical boxes mark two different Hadoop stages, a Map and a Reduce stage. One may see that, even though the Hadoop stages are different, the workload traces are similar. Both show high Memory usage and high Net usage. The predicted regime captures this resemblance on the workload trace.

as Machine Learning, SQL-query based, usual User Defined Functions for databases, and Natural Language Processing workloads.

The goal of this experiment is to validate the methodology for different workloads. For that purpose, we use dataset B (see Section V). In this particular case, we used 10GB data scale samples of the 30 TPCx-BB jobs. For the learning process we keep the same hyper-parameters from the previous experiment.

Figure 7 shows the phases predicted for three of the new workloads: a Natural Language Processing (TPCx-BB query 19), an SQL-query based workload (TPCx-BB query 14) and a Machine Learning workload (TPCx-BB query 20). As it can be seen, similar to previous experiments different learned regimes capture characteristic patterns that are consistent along workload traces. This set of experiments show that the pipeline can be used not only in Hadoop traces, but also in other types of workloads. The results provide learned regimes that match the differenced behaviors that we would expect when looking at the workload traces.

G. Discussion on Portability of the Model Across Workloads

A model trained on a specific type of workload might not be suitable for use on another. This could be because the data may be quite different in shape as well as in feature ranges. For example, our experiments used CPU data with values in the zero to 100 *(number of cores) range. If all the training data contains workloads executed in a single core machine, then all the phases will be learned in that range. Therefore, if a new trace appears taking values outside that range, the model may give unusable phase results. It is important to determine the range of the features of the production/test data at which we aim to apply the method.

H. Exp6: Validating the Method Against a Classical Phase Detection Benchmark

To further validate the phase detection method, we have used it to predict phases in human motion data from Hsu *et al.* [38], a well known dataset used to validate learning of multi-dimensional time-series. The data contains time-series with information concerning humans performing different movements. The time-series values correspond to measurements of body parts; for example, one of the dimensions of the data corresponds to the axis-angle rotation of the pelvis joint. We have prepared a couple of tests involving different motion styles, to show that the method is able to detect different behaviors from different kinds of time-series. Both tests

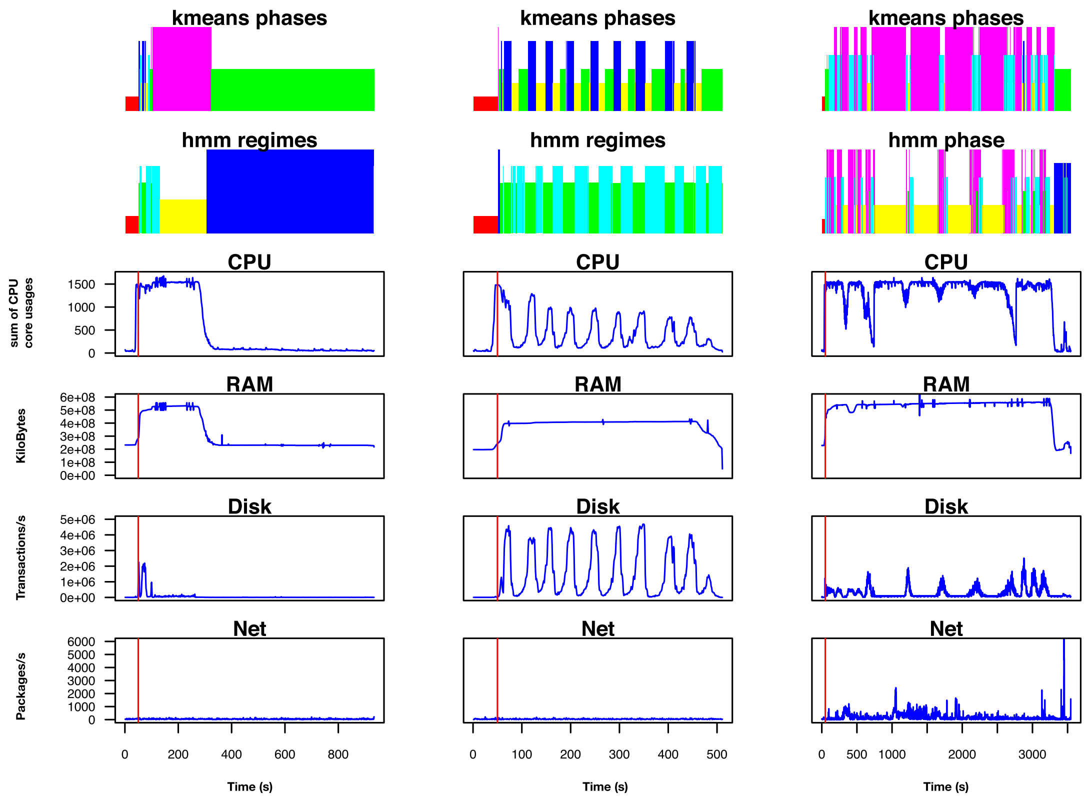


Fig. 7. Three different applications and the predicted phases. From left to right: a Natural Language Processing workload, a SQL query and a Machine Learning Workload. Notice that they also represent a combination of different time-scales, from relatively short jobs in the range of few minutes, to a job that takes around one hour to complete.

use the original data, which contains 108 features per time step.

The first experiment illustrates the importance of the learned representation given by the CRBM. We have taken two sequences of length 2000 from the dataset, one containing walking traces and the other jogging traces. We have concatenated the sequences to create a single example of length 4000. Then we have trained 30 k-means models (with different random initializations); 15 models use the original data and the other 15 use the processed data by the CRBM. Figure 8 shows the results of the phases given by the 30 models. The top 6 outcomes correspond to the different results of the 15 k-means trained with the original data. The bottom two outcomes correspond to the different results of the other 15 models. Notice that while raw data (six top series) produces inconclusive results, passing data through the CRBM allows k-means to discover a single stable pattern. The CRBM version produces two patterns which are actually the same if we flip the labels. Moreover, these two solutions match the walking and the jogging phases with some mixing around time step 3000.

For the second experiment, we have selected four traces of length 500 containing "walking" at slow/normal speed and "jogging" at slow/normal speed. Then we have concatenated the traces to create a single sequence of length 2000. We trained several times with different random initializations 3 types of pipelines. The first pipeline is a simple k-means using the original trace. The second pipeline is a CRBM followed by a k-means. The third pipeline is a CRBM followed by an HMM.

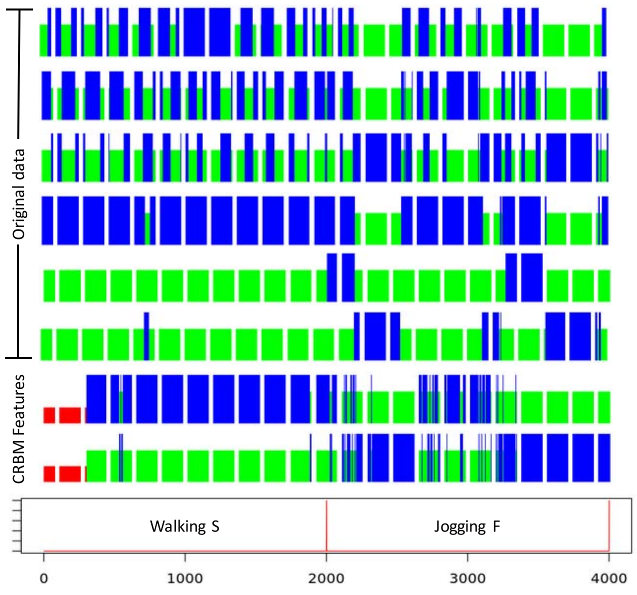


Fig. 8. Results for different random initialization of k-means.

The first sequence in Figure 9 shows one of the several possible solutions of the k-means. As in the previous experiment, the results depend greatly on the random initialization. The second sequence shows one of the two possible outputs given

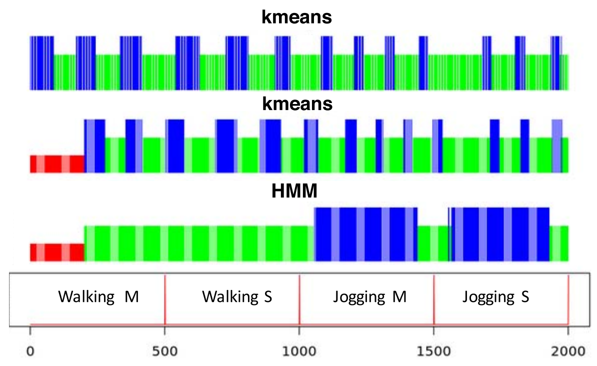


Fig. 9. The lower plot shows the different tags for each of the four concatenated multidimensional time-series. The tags are Jogging and Walking. Both tags contain two different speeds: medium (M) and slow (S).

by the CRBM k-means pipeline (the other is the same with the labels flipped). The third sequence shows one of the two solutions given by the CRBM-HMM pipeline (and again the other is the same with the labels flipped).

We can see that the CRBM-HMM pipeline is able to correctly differentiate the walking phase from the jogging phase, with some error around position 1500, where the trace behavior changes from jogging at middle speed to jogging at slow speed. Nevertheless the CRBM k-means pipeline proposes a phase that does not match the label.

VII. Conclusion

In this work, we present a method for modeling and discovering phases in time-series in an unsupervised way, by using Conditional Restricted Boltzmann Machines to encode n_v dimensional feature input vectors into n_h dimensional vectors, taking the time dimension into account, and feeding them to Hidden Markov Models. We understand as "phases" periods of time displaying similar behaviors.

Workload profiling and resource consumption phase detection are very relevant problems in the areas such as High Performance Computing and Cloud Computing. For this reason, we validated the approach on a couple of datasets containing traces from application executions on data-centers: One dataset containing executions traces of Apache Hadoop jobs and the other dataset containing Spark jobs. Such a scenario implies multi-dimensional time-series data, without either clear labels or clear expert methods for automatically identifying phases. The proposed approach does not require feature engineering, so it can be easily automated, thereby helping decision systems when applications become more complex. Moreover, we find no reason to consider that this method can not also be used for other similar scenarios with time-series.

To verify the validity of the phases, we have presented some sequential performance data to the model, such as workload traces from the ALOJA dataset as a case of use towards data-center management and application characterization. The model is able to generate phases that, upon careful examination on the workload traces, separate different behaviours found in the telemetry traces. Further, as a known case towards a sanity check, the *Motion* dataset used for evaluating time-series. The proposed approach is able to identify distinct behaviors in both cases. In the principal case for the workload traces, we are able to verify that the proposed phases capture different properties from the workloads, consistently characterizing executions by resource consumption for different kinds of application. In the case of the Motion data we are able to show that the pipeline would differentiate walking from jogging traces.

From the experimental results, we have find that CRBMs plus clustering algorithms are able to discover phases on different workload executions, each one corresponding to a specific resource usage pattern. Given that one of the used datasets corresponded to Hadoop executions, we are able to compare the discovered phases with the different Hadoop stages, with the observation that different Hadoop workloads have different behaviors on same phases. This enable us to identify characteristic patterns not only for complete executions, but also for parts of an execution. This method also allows us to generate automatically a fingerprint for applications which can be used to identify them.

We observed that Hidden Markov Models tend to yield more robust results when compared to k-means, making phase prediction less sensitive to noise than k-means (e.g., k-means switch phases back and forth when a resource produces isolated peaks). This behavior is probably a consequence of the HMM, taking into account previous phase values when predicting the next phase value.

Further improvements for this work include the addition of new descriptive variables from the workload traces, describing the environment where applications are being executed. Such additions will provide information about the performance capabilities, allowing us to describe not only execution phases but also status phases. Furthermore, a window mechanism could be implemented on the method output to prevent hysteresis effects when two candidate phases repeatedly switch in a brief period of time, thereby providing greater robustness to the solution towards decision-making in application scheduling. Finally, in scenarios where data is tagged with relevant information, the method can be expanded by adding new components at the output. For example, supervised learning methods could be used to classify applications.

References

[1] D. Magalhães, R. N. Calheiros, R. Buyya, and D. G. Gomes, "Workload modeling for resource usage analysis and simulation in cloud computing," *Comput. Elect. Eng.*, vol. 47, pp. 69–81, Oct. 2015.

[2] S. Esfandiarpoor, A. Pahlavan, and M. Goudarzi, "Structure-aware online virtual machine consolidation for datacenter energy improvement in cloud computing," *Comput. Elect. Eng.*, vol. 42, pp. 74–89, Feb. 2015.

[3] S. J. Tarsa, A. P. Kumar, and H. T. Kung, "Workload prediction for adaptive power scaling using deep learning," in *Proc. IEEE Int. Conf. IC Design Technol.*, Austin, TX, USA, May 2014, pp. 1–5.

[4] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner, "JustRunit: Experiment-based management of virtualized data centers," in *Proc. Conf. USENIX Annu. Tech. Conf. (USENIX)*, San Diego, CA, USA, 2009, p. 18.

[5] R. Thonangi, V. Thummala, and S. Babu, "Finding good configurations in high-dimensional spaces: Doing more with less," in *Proc. IEEE Int. Symp. Model. Anal. Simulat. Comput. Telecommun. Syst.*, Baltimore, MD, USA, Sep. 2008, pp. 1–10.

[6] A. Thusoo *et al.*, "Data warehousing and analytics infrastructure at Facebook," in *Proc. ACM SIGMOD Int. Conf. Manag. Data (SIGMOD)*, Indianapolis, IN, USA, 2010, pp. 1013–1020.

[7] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic resource inference and allocation for mapreduce environments," in *Proc. 8th ACM Int. Conf. Auton. Comput. (ICAC)*, Karlsruhe, Germany, 2011, pp. 235–244.

[8] M. Otte and S. Richardson, "An HMM applied to semi-online program phase analysis," Dept. Comput. Sci., Univ. Colorado at Boulder, Boulder, CO, USA, Rep. CU-CS-1034-07, 2007.

[9] P. Nagpurkar, C. Krintz, M. Hind, P. F. Sweeney, and V. T. Rajan, "Online phase detection algorithms," in *Proc. Int. Symp. Code Gener. Optim. (CGO)*, New York, NY, USA, 2006, pp. 111–123.

[10] G. W. Taylor, G. E. Hinton, and S. T. Roweis, "Modeling human motion using binary latent variables," in *Advances in Neural Information Processing Systems 19*, P. B. Schölkopf, J. C. Platt, and T. Hoffman, Eds. Cambridge, MA, USA: MIT Press, 2007, pp. 1345–1352.

[11] G. W. Taylor and G. E. Hinton, "Factored conditional restricted Boltzmann machines for modeling motion style," in *Proc. 26th Int. Conf. Mach. Learn. (ICML)*, Montreal, QC, Canada, 2009, pp. 1025–1032.

[12] N. Mishra, J. D. Lafferty, and H. Hoffmann, "ESP: A machine learning approach to predicting application interference," in *Proc. IEEE Int. Conf. Auton. Comput. (ICAC)*, Columbus, OH, USA, Jul. 2017, pp. 125–134.

[13] C. Delimitrou and C. Kozyrakis, "iBench: Quantifying interference for datacenter applications," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Portland, OR, USA, Sep. 2013, pp. 23–33.

[14] Y. Zhu *et al.*, "Packet-level telemetry in large datacenter networks," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 479–491, Aug. 2015.

[15] *Netflix Atlas*, Netflix, Los Gatos, CA, USA, 2017.

[16] J. Polo *et al.*, "Deadline-based mapreduce workload management," *IEEE Trans. Netw. Service Manag.*, vol. 10, no. 2, pp. 231–244, Jun. 2013.

[17] J. Polo *et al.*, "Resource-aware adaptive scheduling for mapreduce clusters," in *Proc. 12th ACM/IFIP/USENIX Int. Conf. Middleware*, Lisbon, Portugal, 2011, pp. 187–207.

[18] J. Liu, Y. Zhang, Y. Zhou, D. Zhang, and H. Liu, "Aggressive resource provisioning for ensuring QoS in virtualized environments," *IEEE Trans. Cloud Comput.*, vol. 3, no. 2, pp. 119–131, Apr./Jun. 2015.

[19] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. OSDI*, San Francisco, CA, USA, Dec. 2004, pp. 137–150.

[20] *Hadoop*, Apache Softw. Found., Forest Hill, MD, USA, Dec. 2017. [Online]. Available: https://hadoop.apache.org

[21] U. Sirin, P. Tözün, D. Porobic, and A. Ailamaki, "Micro-architectural analysis of in-memory OLTP," in *Proc. Int. Conf. Manag. Data (SIGMOD)*, San Francisco, CA, USA, 2016, pp. 387–402.

[22] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A portable programming interface for performance evaluation on modern processors," *Int. J. High Perform. Comput. Appl.*, vol. 14, no. 3, pp. 189–204, Aug. 2000.

[23] A. Awan, M. Brorsson, V. Vlassov, and E. Ayguade, "Performance characterization of in-memory data analytics on a modern cloud server," in *Proc. Int. Conf. Big Data Cloud Comput.*, Dalian, China, Aug. 2015, pp. 1–8.

[24] M. Amaral, J. Polo, D. Carrera, S. Seelam, and M. Steinder, "Topology-aware GPU scheduling for learning workloads in cloud environments," in *Proc. Int. Conf. High Perform. Comput. Netw. Stor. Anal. (SC)*, Denver, CO, USA, 2017, pp. 1–17.

[25] N. Rameshan, L. Navarro, E. Monte, and V. Vlassov, "Stay-away, protecting sensitive applications from performance interference," in *Proc. 15th Int. Middleware Conf. (Middleware)*, Bordeaux, France, Dec. 2014, pp. 301–312.

[26] A. Rosa, W. Binder, L. Y. Chen, M. Gribaudo, and G. Serazzi, "ParSim: A tool for workload modeling and reproduction of parallel applications," in *Proc. IEEE 22nd Int. Symp. Model. Anal. Simulat. Comput. Telecommun. Syst.*, Paris, France, Sep. 2014, pp. 494–497.

[27] C. Ding, S. Dwarkadas, M. C. Huang, K. Shen, and J. B. Carter, "Program phase detection and exploitation," in *Proc. IPDPS*, 2006, p. 8.

[28] P. Pipada *et al.*, "LoadIQ: Learning to identify workload phases from a live storage trace," in *Proc. 4th USENIX Workshop Hot Topics Stor. File Syst.*, Boston, MA, USA, Jun. 2012, p. 3.

[29] C. Sammut and G. I. Webb, Eds., *Baum-Welch Algorithm. Encyclopedia of Machine Learning*. Boston, MA, USA: Springer, 2010.

[30] X. Cai and X. Lin, "Forecasting high dimensional volatility using conditional restricted Boltzmann machine on GPU," in *Proc. IPDPS Workshops*, Shanghai, China, 2012, pp. 1979–1986.

[31] R. Salakhutdinov and G. Hinton, "Using deep belief nets to learn covariance kernels for Gaussian processes," in *Proc. 20th Adv. Neural Inf. Process. Syst.*, 2008, pp. 1–8.

[32] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, Aug. 2002.

[33] G. D. Forney, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, no. 3, pp. 268–278, Mar. 1973.

[34] J. L. Berral *et al.*, "ALOJA-ML: A framework for automating characterization and knowledge discovery in Hadoop deployments," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, Sydney, NSW, Australia, 2015, pp. 1701–1710.

[35] T. Rabl *et al.*, "Big data benchmarking," in *Proc. 7th Int. Workshop (WBDB)*, vol. 10044. New Delhi, India, 2015, Dec. 2016, pp. 71–84.

[36] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *Proc. 22nd Int. Conf. Data Eng. Workshops*, Long Beach, CA, USA, 2010, pp. 41–51.

[37] A. Ghazal *et al.*, "Bigbench: Towards an industry standard benchmark for big data analytics," in *Proc. ACM SIGMOD Int. Conf. Manag. Data (SIGMOD)*, New York, NY, USA, 2013, pp. 1197–1208.

[38] E. Hsu, K. Pulli, and J. Popović, "Style translation for human motion," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1082–1089, Jul. 2005.

David Buchaca Prats received the degree in mathematics from University of Barcelona in 2012 and the M.Sc. degree in artificial intelligence from BarcelonaTech-UPC in 2014. He is currently pursuing the Ph.D. degree with the Data-Centric Computing, Barcelona Supercomputing Center. He is an applied mathematician, working in applications of artificial neural networks.

Josep Lluís Berral (M'18) received the degree in informatics in 2007, the M.Sc. degree in computer architecture in 2008, and the Ph.D. degree from BarcelonaTech-UPC, computer science in 2013. He is a Data Scientist, working in applications of data mining and machine learning on data-center and cloud environments with Data-Centric Computing, Barcelona Supercomputing Center. He was with the High Performance Computing Group, Computer Architecture Department-UPC, and the Relational Algorithms, Complexity and Learning Group, Computer Science Department-UPC.

David Carrera (M'08) received the M.S. and Ph.D. degrees from BarcelonaTech-UPC in 2002 and 2008, respectively. He is an Associate Professor with the Computer Architecture Department, UPC. He is an Associate Researcher with the Data-Centric Computing, Barcelona Supercomputing Center. His research interests are focused on the performance management of data center workloads. He has been involved in several EU and industrial research projects. In 2015, he was awarded an ERC Starting Grant for the project HiEST. He was a recipient of the IBM Faculty Award in 2010.