Information Sciences 433–434 (2018) 381–400

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Self managed virtual machine scheduling in Cloud systems

Stelios Sotiriadis[a,*], Nik Bessis[b], Rajkumar Buyya[c]

[a] The Edward Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Bahen Centre for Information Technology St. George Campus, 40, Toronto, ON M5S 2E4, Canada
[b] Department of Computer Science, Edge Hill University, St Helens Rd, Ormskirk, Lancashire L39 4QP, UK
[c] Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, University of Melbourne, Parkville VIC 3010, Australia

A R T I C L E   I N F O

Article history:
Received 29 March 2016
Revised 14 June 2017
Accepted 6 July 2017
Available online 8 July 2017

Keywords:
Cloud computing
OpenStack
Virtual machine placement
Virtual machine scheduling

A B S T R A C T

In Cloud systems, Virtual Machines (VMs) are scheduled to hosts according to their instant resource usage (e.g. to hosts with most available RAM) without considering their overall and long-term utilization. Also, in many cases, the scheduling and placement processes are computational expensive and affect performance of deployed VMs. In this work, we present a Cloud VM scheduling algorithm that takes into account already running VM resource usage over time by analyzing past VM utilization levels in order to schedule VMs by optimizing performance. We observe that Cloud management processes, like VM placement, affect already deployed systems (for example this could involve throughput drop in a database cluster), so we aim to minimize such performance degradation. Moreover, overloaded VMs tend to steal resources (e.g. CPU) from neighbouring VMs, so our work maximizes VMs real CPU utilization. Based on these, we provide an experimental analysis to compare our solution with traditional schedulers used in OpenStack by exploring the behaviour of different NoSQL (MongoDB, Apache Cassandra and Elasticsearch). The results show that our solution refines traditional instant-based physical machine selection as it learns the system behaviour as well as it adapts over time. The analysis is prosperous as for the selected setting we approximately minimize performance degradation by 19% and we maximize CPU real time by 2% when using real world workloads.

1. Introduction

In latest years, Cloud computing has been emerged as one of the most widely used systems for provisioning virtual resources to everyday Internet users. Many cloud platforms host services (including infrastructure, platform and software) that are available on a pay as you go and on-demand model. One of the most important process of such systems is the virtual machine (VM) scheduling that determines the way in which Physical Machine (PM) resources are allocated in order to launch a VM instance (a process called VM placement) [33]. Simple solutions include VM scheduling to a PM that has the most available computational resource (for example RAM) and is within an availability zone (location constraints set by users) so is ranked as the best available host among others. There are various criteria for VM placement such as scheduling to hosts with fewer running instances or according to filtering with regards to current resource utilization levels. Other

* Corresponding author.
E-mail addresses: s.sotiriadis@utoronto.ca, s.sotiriadis@intelligence.tuc.gr, steliosot@msn.com (S. Sotiriadis), Nik.Bessis@edgehill.ac.uk (N. Bessis), rbuyya@unimelb.edu.au (R. Buyya).

http://dx.doi.org/10.1016/j.ins.2017.07.006
0020-0255/© 2017 Elsevier Inc. All rights reserved.

include multi sub-scheduling layers that allow more refined placement according to network traffic, availability zones and other criteria. In general, a second level of scheduling could provide flexibility over the regular scheduling decision [12].

In this work we propose the concept of VM scheduling according to resource monitoring data extracted from past resource utilizations (including PMs and VMs). Our solution applies for Cloud and Inter-Cloud systems as described in [34,37] following the requirements as presented in [2]. Current systems do not support dynamic VM placement, for example OpenStack performs a filtering and weighing of PMs in an instant based way, and does not take into consideration past system behaviour or VMs resource usage. The problem is getting worst when the scale is increased for example in large scale Cloud and inter-cloud platforms [32]. This means that PMs that host idle VMs could be shown as busy during specific time instances, however overall are under-utilized. Another important factor is that Cloud systems use an over-commit resource sharing method [17], where PMs provide more cores and memory than the capacity of the physical host can serve. This is based on the fact that users, have underutilized VMs and do not have the same resource usage pattern over the day. Finally, Cloud management processes, such as VM placement, affect already deployed systems (for example this could involve throughput drop in a database cluster) as well as overloaded VMs tend to steal CPU times from neighbouring VMs. These represent simple cases that demonstrate the need for a more refined VM scheduling that could improve performance.

In this work we focus on the OpenStack platform[1], that is an open source software to build private and public clouds. OpenStack default configuration involves placing VMs by selecting the host with the most available memory until the VMs number exceeds the limit. Such behaviour overloads powerful PMs in the stack and leaves low RAM PMs under-utilized [13]. Similarly in [17], authors suggest that it might be more efficient to launch VMs to idle hosts with powerful CPUs and less memory, thus bypassing the default OpenStack scheduler could improve cluster performance. Having said that, current OpenStack scheduling is considered as simple since it sorts and weighs PMs on instantaneously collected resource usage. It should be mentioned that a more detailed discussion of schedulers is presented in [35]. In this work we suggest real time resource analytics based on past resource usage by developing a machine learning model that analyzes PMs and VMs resource usage on-the-fly. The training data set is populated on a regular interval so the solution provides indications and predictions as vital factors for VM placement. We expect that our solution will provide deeper understanding of parameters affecting performance of the already deployed system. We present an experimental analysis such issues based on real world systems (e.g. Apache Cassandra, MongoDB, Elasticsearch) and workload such as Yahoo! Cloud Serving Benchmark (YCSB).

The primary "contribution" of our paper is a new Cloud VM scheduling algorithm that is dynamic and adaptive based on historical resource usage of PMs and VMs. We perform pattern extraction according to continuous monitoring and data analysis using machine learning. Our contribution is on two cases, (a) we minimize performance degradation of already deployed systems during the VM scheduling process by demonstrating elimination of throughput drops in database systems and (b) we increase CPU real time of deployed VMs by minimizing CPU steal times caused by overloaded PMs. So, in Section 2 we present an extensive literature review analysis on algorithms and approaches based on cloud VM placement and scheduling, in Section 3 we presents the VM scheduling algorithm, in Section 4 we present the performance evaluation and in Section 5 we discuss the conclusions and future research directions.

2. Literature review

This section presents the literature review study for VM placement. We classify accordingly to different areas related with (a) cloud platform scheduling, (b) energy efficiency and power cost management, (c) resource provisioning and (d) optimization of resource usage related parameters.

2.1. Cloud platform scheduling

Cloud platform scheduling refers to solutions that focus on the resource management layer and resource orchestration of the VM placement process.

The work in [25] present an architecture for scheduling strategies based on a broker mechanism. They use different optimization criteria related with performance optimization, user constraints (i.e. VM placement) and environmental conditions (i.e. instance prices). However, this work focuses more on multi cloud, it does not consider past service experiences and it does not utilize machine learning models. In [8], authors present an application profiling method for VM placement and they suggest that the scheduling and resource allocation can be improved. They propose a method based on the canonical correlation analysis to create the relationships between the application performance and resource usage. Furthermore, they correlate the application performance with a canonical weight vector that represents the level of involvement of system factors. The experimental analysis show that high predictions can be achieved on high weights.

In [42], authors focused on the problem of VM scheduling in OpenStack systems and present a cloud CPU resource management system for VM hosting. They include a real time hypervisor, VM scheduler (to allow VMs to share hosts without interfering performance and VM to host mapping strategy. The experimental results show high CPU resource utilization when co-hosting VMs. In [3] authors presented a high availability property for a VM named as *k*-resilient. The work suggested a novel algorithm that guarantees resilience meaning that a VM can be relocated to a non failed host without the

[1] OpenStack: https://www.openstack.org.

need to relocate other VMs. The experimental solution is based on a simulation, and the results show optimization in load balancing compared to a standalone host solution. In [36], we presented a Cloud simulation framework where we included basic Cloud scheduling features.

In [26] authors presented V-Man that is a decentralize algorithm for VM consolidation in clouds. The aim of the algorithm is to maximize the number of PMs by iteratively producing more allocations. The PMs exchange messages in order to maintain an unstructured overlay network in order to exchange VMs so PMs from heave usage nodes to move to low usage ones. They present a peer to peer simulation study and they demonstrate that an optimal allocation can be achieved in a five round of message exchanges. Yet, the authors assume that all VMs are identical.

2.2. Energy efficiency and power cost management

This section presents approaches focusing on VM placement to achieve objectives for energy efficiency for cloud clusters and on more effective cost management.

In [11] the work aims to optimize VM placement and traffic flow routing by presenting it as an optimization problem. They suggest that 10–20% of the total power consumption is provoked by the network elements, thus they propose the "VMPlanner" that includes three approximation algorithms that are (a) the VM grouping according to minimized traffic volume, (b) VM-group to server-rack mapping (for placing VMs into rack more efficiently) and (c) power-aware inter-VM traffic flow routing for minimizing the number of paths in the network. They use Greedy Bin-Packing algorithm, as presented [31], to select the path with the sufficient capacity.

In [22] authors study load placement policies for cooling and datacenter temperatures. They follow the idea of VM placement and migration according to different electricity prices and temperatures during the time. Authors base their system on the assumption that estimated of the running time of jobs is given by the users. They use round robin algorithm and a cost aware static policy for comparing their results. In [1], authors present an analytical study to explore single VM migration and consolidation and their respective online deterministic algorithms. They provide adaptive heuristics (VM placement optimization and power aware best fit decreasing) that analyze historical data of physical machines with regards to resource usage for optimizing energy and performance of VMs. They evaluate the algorithms using a large scale experimental simulation.

In [39] authors explore the service-oriented VM placement as an optimization problem. They try to solve the problem using a graph of Tree (to minimize communication between VMs) and Forest algorithm (for balancing traffic load between VMs). To evaluate the algorithms they present a comparison with the Best Fit algorithm. They claim that the Forest algorithm decreases outbound communication cost by 22% and Tree algorithm by 92%. They authors suggest that dynamic VM allocation will be further investigated. In [5] present VM placement and consolidation of VMs in cloud datacenters using min-max. The authors suggest that guided by application utilities could provide better resource allocation, so high utility applications get most resources. They experiment using synthetic and real datacenter testbed and they conclude that the PowerExpandMinMax algorithm is the best utility "for power-performance tradeoffs in modern data centers running heterogeneous applications".

In [41], authors present a VM placement framework for minimizing energy consumption and maximize profits. They suggest that these are constraint satisfaction problems and they suggest a utility function that expresses the SLA satisfaction. From the perspective of the system they provide a VM placement formulation in order to maximize the number of the machines to be turned off. The experimental analysis is based on the Xen hypervisor and authors demonstrate that different parameters can affect the operational costs and to balance the QoS.

In [15] authors present a resource allocation problem that aims to minimize the total energy cost of cloud system, in a probabilistic way. Their algorithm places VMs to PMs using dynamic programming and convex optimization. Decision epochs are used in order to estimate resource requirements. The algorithm is evaluated using simulations and results shows a VM placement that minimizes the power cost. In [16] authors try to solve cloud energy-efficient VM placement by creating multiple copies of VMs and placed them into PMs using local search. The experimental analysis shows 20% improvement in energy consumption compared with selected heuristics.

In [9] authors provide a framework that allows allocation of VMs in a datacenter to achieve energy awareness. They further decouple constraints from algorithms and they implement 16 frequently used SLA parameters in the form of constraints. The experimental analysis shows an 18% improvement in savings of both energy and CO_2 emissions. In [23], the work presents the EnaCloud that is an energy aware heuristic algorithm for VM placement in a dynamic way by considering energy efficiency. Authors present an experimental case for Xen VMM and they claim that their solution saves energy in cloud platforms by comparing their algorithm with FirstFit and BestFit. Also, they separate their workloads to web/database server, compute-intensive and common applications.

2.3. Resource provisioning

This section summarizes the resource provisioning VM placement approaches.

In [6], authors suggest that in cloud computing there are two provisioning plans, the reservation and the on-demand plan for cloud consumers. However, advance reservation is difficult to be achieved because of the uncertainty of the resource usage by the users. To reduce this problem, authors suggest an optimal cloud resource provisioning algorithm based on a stochastic programming model. The work is based on an numerical analysis and evaluations is based on simulations that show minimization of on-demand costs. Also, they suggest that VM outsourcing i.e. private to public cloud could offer significant advantages. In [7], authors focused on the VM consolidation problem in OpenStack systems related with features such as power, CPU and network sharing. They propose a cloud management platform to optimize these features. They conclude that VM consolidation among PMs may offer significant benefits however it can also lead to significant performance side effects.

In [40], authors proposed an autonomic resource management component that decouples provisioning of resources from the dynamic placement of virtual machines. They introduce a utility function that optimizes VM provisioning based on constraint satisfaction problems. In [38] a cloud brokering mechanism for VM placement in multiple clouds is presented. The criteria include price, performance, hardware and load balancing. In this work users define their price and their minimum performance and the algorithms place VMs accordingly. They evaluate their solution using a high throughput cluster deployments across cloud providers. They suggest that they achieve 20% better load balancing with low performance degradation.

2.4. VM placement for optimizing resource usage

This section summarizes the VM placement approaches that aim to achieve objective or multi-objective parameters such as optimization of resource usage for more efficient utilization of the PMs within a cloud cluster.

In [14], authors discusses the concept of VM placement from an analytical perspective of VM cloud placement to achieve power efficiency and resource utilization. They present the VM placement as an ant colony optimization problem in order to minimize resource wastage and power consumption based on non dominated solutions. They use PM resource usage statistics for a period of time. In [21] authors focus on the prediction of the effect that could have critical parameters on the performance of VMs. These are allocation percentages, real-time scheduling and co-placement of VMs in the same PM. They use linear regression and genetically optimized artificial neural networks for measuring the prediction degradation. They suggest that an interest aspect of future work is the detection of the workload types for applications that will affect the predicted value.

In [28], authors present an objective method that can be used to compare VM placement algorithm. The response variables are the user experience, resource utilization, types of VMs and other. Authors present an extensive comparison of 18 algorithms for VM placement and conclude that the choice of the cluster influences more than the selection of nodes. In [10], authors present a technology-neutral interfaces for federated cloud systems aiming to VM placement, migration and monitoring. They base their design in a grid computing monitoring architecture and they provide algorithms to demonstrate cross VM site management. In [27], authors focused on optimizing networking scalability by proposing traffic aware VM placement. They suggest that better placement will can offer improved communication by limiting the distance between them. They present an optimization problem and they design a tow tier approximation algorithm that is based on traffic patterns to achieve better traffic aware VM placement. They provide the cluster-and-cut algorithm (that partitions VMs to clusters) and the VMMinKcut (partitioning VMs to clusters with minimum inter-cluster traffic). The experimental analysis is prosperous and the algorithm over-performs by 10% the selected benchmarks.

In [30], suggested that VM placement and migration can be executed during unexpected network latencies in a more sophisticated way that minimizes the data transfer times. They propose an analytical model with algorithms for both cases that places VMs to PMs by considering the network conditions among PMs and data storage. Also they present a simulation, where results are prosperous for selected configurations. In [43], authors presented the VM placement as a multi-objective optimization problem that aims to minimize the total resource wastage, power consumption and thermal dissipation costs. They use genetic algorithms to search using conflicting objectives, i.e. for VM placement the objectives are performance, scalability and robustness, and every VM placement has a corresponding chromosome. The results are prosperous and achieve significant optimization in performance when compared with he bin-packaging algorithm and single-objective approaches.

In [18], a structural constraint- aware VM placement is presented that include demand, communication and availability. They present an optimization problem and they design a hierarchical placement approach using algorithms including a constraint-aware VM grouping (with minimal communication cost) and VM groups to server rack assignment. They simulate different VM placement settings and they optimize the communication cost during this action. In [19] authors present a VM placement problem to minimize the traffic cost. They provide a Markov approximation solution based on an online algorithm that is dynamic in terms of changing traffic loads that minimizes the number of VMs that need to be relocated. Also, they provide a performance evaluation by comparing server placement approaches (sequential and random) and routing selection approaches (shortest path and oblivious) and results are prosperous since their algorithm provides significant improvements in large and small flows.

In [4], authors present the Min Cut Ratio-aware VM Placement algorithm that aims to a placement that satisfies communication and resilience to demand time variations. The experimental analysis show improved datacenter scalability and reduces the number of dropped packaged by supporting time varying traffic demands. In [24] authors present the least-busy VM placement algorithm for dynamic scalling performance optimization of service oriented applications hosted in clouds. They present an experimental analysis comparing their solution with round robin and they observe an a 2% to 3% fewer VMs that achieve 12% to 16% average improvement for VM placement.

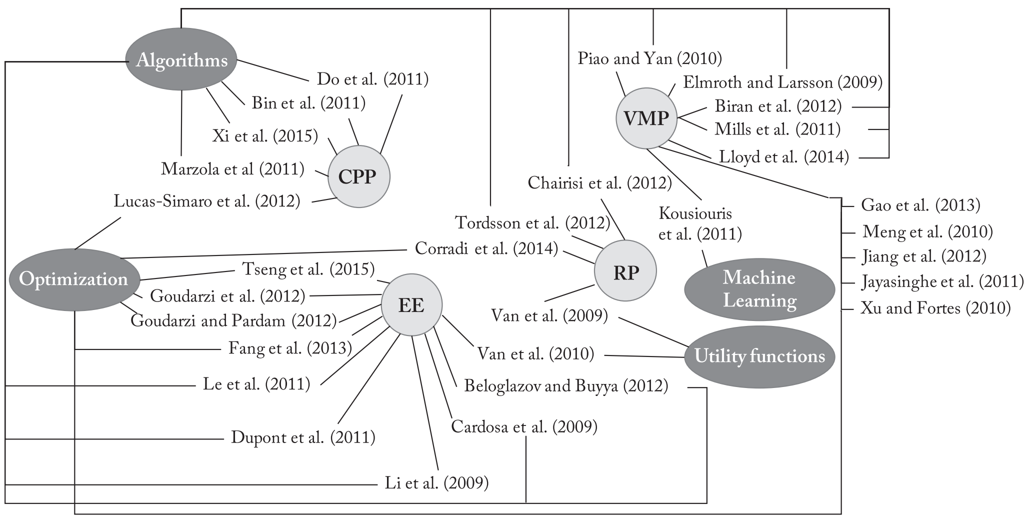

Fig. 1. Classification of literature review approached according to CPP, EE, VMP and RP and according to their method (algorithms, optimization, utility functions and machine learning).

2.5. OpenStack scheduling

OpenStack is a open source platform for private and public clouds [29]. It uses the nova-scheduler to decide how VMs should be placed among the PMs of the OpenStack cluster. Also, it does a systematic search for the best resource by having an aggregated resource view of all hosts in the cluster [20]. It follows the concept of filtering, meaning that VMs are placed according to the PMs parameters such as computational resources (CPU, memory etc.), architectural characteristics (hypervisors, image properties etc.) and availability properties (i.e. zones). According to [20] these are classified as simple, chance and zone using a host ranking weight filtering. Moreover, the scheduler stores available hosts into a list and updates it at intervals configured by the cloud administrator. At the time of the VM scheduling it makes informed decisions based on the following:

1. A filtering algorithm for PMs to decide which are capable for hosting the required virtual resources (accepted or rejected). There are many available filters like the Aggregate Core Filter based on CPU allocation ratios, Aggregate Disk Filter based on disk allocation ratios, Aggregate Image Properties Isolation for determine images that are matched with aggregation metadata and many others as presented in [29].
2. A weighting algorithm to decide which host from the filtered list is the most dominant for the current request (i.e. by default is based on RAM weighting).

OpenStack uses two approached to achieve better resource utilization according to [17] as follows:

1. *Over-commit* allows CPU and RAM sharing by VMs meaning that OpenStack commits more resources that could actually provide by the physical host to an over-commit limit, thus more users could be served by a PM. The idea is based on the fact that usually users do not simultaneously use their resources at highest levels (an action called resource pegging).
2. *Scheduling improvement* allows the OpenStack administrator to configure scheduling algorithms according to the PMs resource usage. This includes the filtering and weighing of VMS as presented before.

An interesting configuration in the OpenStack scheduler is the JSON filter that allows simple JSON based grammars for selecting hosts. This could include a further step of the selection process i.e. after filtering and weighting to allow a more sophisticated querying combing various metrics to optimize scheduling. In particular schedulers does not only affect the performance from the perspective on where to place the VM but also influence the performance of the hosts in terms of over-committing limit. This means that VMs are placed without considering their projected resource usage levels at it is impossible to forecast such behaviour before the actual deployment.

2.6. Review conclusions

Fig. 1 presents the approaches for Cloud Platform Scheduling (CPP), Energy Efficiency (EE), Resource Provisioning (RP) and VM Placement (VMP). We also correlate the approaches to their methodology including algorithms, optimization, utility functions and machine learning. We can observe that

We also summarize literature review analysis in Appendix Tables 1 and 2. Most of the works like [1,3–7,16,17,41] focus on algorithms and optimization methodologies to achieve better VM placement in cloud systems using either current resource

Table 1
Approaches for Cloud Platform Scheduling (CPP) and Energy Efficiency (EE).

Class	Authors	Method	Method concept	Parameter 1
CPP	Do et al. [8],	Algorithms	Canonical correlation analysis	Scheduling
CPP	Bin et al. [3]	Algorithms	k-resilient	Load balancing
CPP	Xi et al. [42]	Algorithms	Mapping strategy	Resource utilization
CPP	Marzolla et al. [26]	Algorithms	Message exchanging	Resource allocation
CPP	Lucas-Simarro et al. [25]	Optimization	Constraints satisfaction	Resource utilization
EE	Dupont et al. [9]	Algorihtms	Constraints satisfaction	Energy consumption, CO_2
EE	Le et al. [22]	Algorihtms	Heuristics	Electricity price, Temperature
EE	Beloglazov and Buyya [1]	Algorihtms	Heuristics	Resource usage
EE	Li et al. [23]	Algorithms	Heuristics (FirstFit/BestFit)	Energy consumption
EE	Cardosa et al. [5]	Algorithms	Min-max	Power performance
EE	Fang et al. [11]	Optimization	Greedy Bin-Packing	Traffic volume
EE	Goudarzi and Pedram [16]	Optimization	Local search	Energy consumption
EE	Goudarzi et al. [15]	Optimization	Probabilistic Analysis	Resource usage
EE	Tseng et al. [39]	Optimization	Tree & Forest	Communication cost
EE	Van et al. [41]	Utility function	Constraints satisfaction	Energy consumption, profit

Table 2
Approaches for Resource Provisioning (RP) and VM Placement (VMP).

Class	Authors	Method	Method concept	Parameter 1
RP	Tordsson et al. [38]	Algorihtms	Cloud brokering	Load balancing
RP	Chaisiri et al. [6]	Algorihtms	Stochastic programming model	On-demand costs
RP	Corradi et al. [7]	Optimization	Optimize resource usage	Resource usage
RP	Van et al. [40]	Utility function	Constraints satisfaction	Resource usage
VMP	Piao and Yan [30]	Algorithms	Analytical model	Network factors
VMP	Elmroth and Larsson [10]	Algorithms	Cross VM management	Resource utilization
VMP	Biran et al. [4]	Algorithms	Min Cut Ratio-aware	Communication cost, Resilience
VMP	Mills et al. [28]	Algorithms	Objective method	Resource utilization VM type
VMP	Lloyd et al. [24]	Algorithms	Services	Scalability
VMP	Kousiouris et al. [21]	Machine learning /Algorithms	Linear regression/ Genetic algorithms	Prediction degradation
VMP	Gao et al. [14]	Optimization	Ant colony optimization	Resource wastage, performance
VMP	Meng et al. [27]	Optimization	Approximation	Communication cost
VMP	Jiang et al. [19]	Optimization	Approximation (Markov)	Traffic cost
VMP	Jayasinghe et al. [18]	Optimization	Constraints satisfaction	Server rack assignment
VMP	Xu and Fortes [43]	Optimization	Multi-objective	Scalability, Robustness

usage or historical data from cluster PMs. Works like [9,18,25,40] focus on constraints satisfaction objectives following PMs resource usage. Approaches like [14,23,39] focus on optimizing NP hard VM placement. Approaches like [11,18,19,27] focus on networking aspects in order to optimize the communication cost function usually related with optimal VM placement in order VMs to be close to each other.

The work of [21] is different from aforementioned solutions as approaches the problem from a different perspective as it includes a machine learning linear regression model for predicting behaviour of unknown applications. However, the authors define the VM placement as a heuristic optimization problem using genetic algorithms for quantifying and predicting the performance of the applications.

Based on this discussion, we conclude that to the best of our knowledge, current works does not consider dynamic VM placements according to past VM resource utilization. Yet, almost all the approaches focus on the problem of allocating VMs considering only the PMs resource usage (either real-time, opportunistic or as an optimization problem). Moreover, machine learning models are hardly used in literature.

Cloud platforms use simple weighting schedulers based only on PM resource utilization. Authors in [17] suggest that “the key problem is that the current weighting strategy is weak and leads to inefficient usage of resource”. We are motivated by this work and from the statement as authors further suggest that static and dynamic system usage statistics are vital for calculating the VM placement weight. For example default OpenStack scheduler has an aggregated view of resources and places VMs on large memory systems until the VMs number exceeds the limit, thus leaving low memory systems under-utilized or idles [13]. This overloads powerful PMs in the stack and leaves low RAM PMs under-utilized. Next section presents the VM scheduling algorithm based on already VMs resource utilization.

3. VM scheduling algorithm

This section presents the VM scheduling algorithm that includes two optimization schemes based on machine learning. The algorithm enhances the VM selection phase based on real time monitoring data collections and analysis of physical and virtual resources. Our aim is to strengthen VM scheduling in order to incorporate criteria related to the actual VM utilization levels, so VMs can be placed by minimizing the penalization of overall performance levels. The optimization schemes

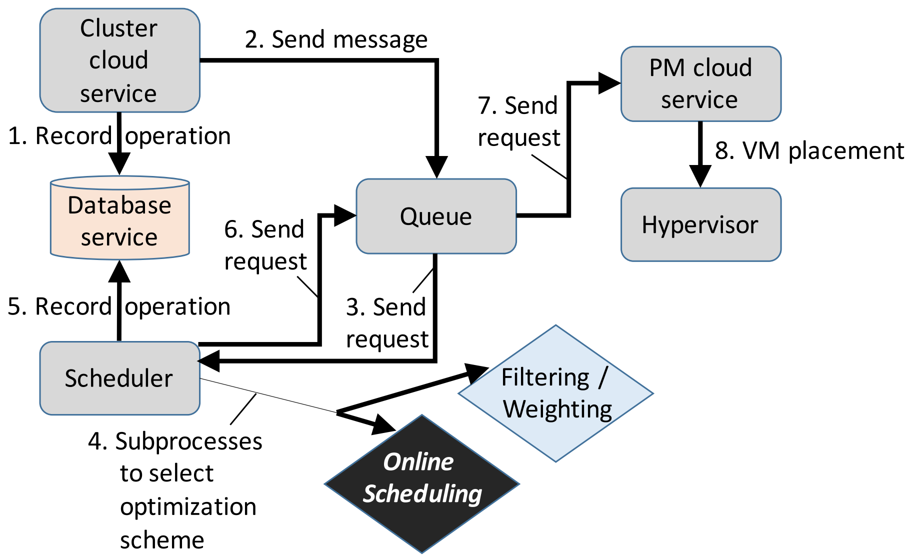

Fig. 2. Online VM Scheduler in OpenStack system.

involve analytics to the already deployed VMs to include (a) maximization of utilization levels and (b) minimization of the performance drops. In Section 3.1 we present the proposed VM scheduler and in Section 3.2 we present the optimization schemes for classifying resources according to past historical records.

3.1. Online scheduler

Our technique is based on the OpenStack VM scheduling process that includes PMs evaluation prior selecting the best host for VM placement. The assumption is that requests for VMs are submitted to a Cloud service (in OpenStack is the nova service) that initiates that process. After PMs evaluation, the VM is scheduled to the best ranked host. In our case, we extend the selection steps to include the online VM scheduling that includes past history records. In detail, we include one more step that allows selected PMs to be sorted according to the ones that (a) have computational resources and (b) will have the least affection to the already deployed systems. to achieve (b) we perform prediction on real time according to the past resource usage. Fig. 2 demonstrates the VM scheduling process.

The process includes 8 steps (starting from receiving the request to placing the VM to the best ranked PM) and are as follows.

Step 1. The cloud hosting service (that is the Cloud fabric controller) receives a request for creating a new VM including resources to be allocated such as number of Cores, memory and hard disk resources. Other information includes network and VM metadata configurations that are out of the scope of this study.

Step 2. The cloud hosting service sends a request to the Cloud database service that records all operations.

Step 3. The cloud hosting service sends a request to a scheduling queue in order to select a PM for scheduling the VM.

Step 4. The scheduler performs the following two subprocesses.

(a) First it collects a list with all available PMs in the cluster and performs a filtering by selecting PMs according to their current available resources. The filter is binary, meaning that a PM is capable or not for hosting a VM. For example if the VM requests 8GM of RAM, all PMs that have greater or equal available RAM will be selected.

(b) Secondly, the online scheduling process comes to enhance the filtering and weighting scheduling by including more refined optimization criteria that are related with the past resource usage over time. This process triggers machine learning classification or regression processes according to the selecting criteria. This process enhances the traditional approach as it includes data analytics from a real time resource monitoring engine and a machine learning algorithm to classify resource usage, instead of based on a simple sorting algorithm. Formula (1) [29] demonstrates the weighting function, where w is the weigher of a host, $wmul$ the weigher multiplier (that is a coefficient of the selected parameter i.e. RAM 1 and CPU -1), n is the normalization parameter and ν is the total number of weighers.

$$w = w_1 mul * n(w_1) + w_2 mul * n(w_2) + \ldots + w_\nu mul * n(w_\nu) \quad (1)$$

In our case we replace this process by using the optimization schemes. In other words, weighing is based on real time data and historical records instead of instantaneous collected PM data. In detail, the algorithm evaluates past

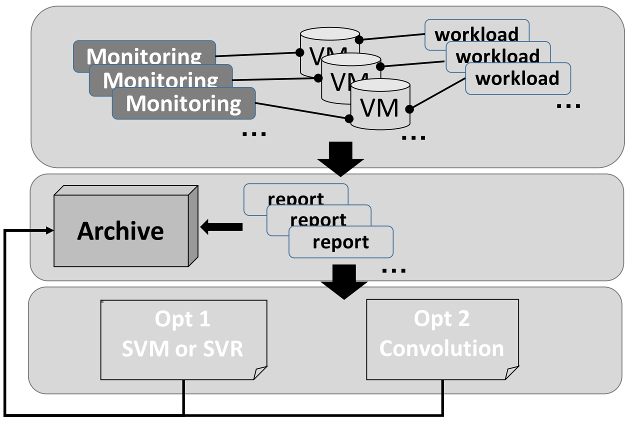

Fig. 3. VM resource monitoring process for online VM scheduling.

resource utilization levels (e.g. last 7 days) and classifies according to the overall resource usage. At the end the list of candidate hosts is populated and the resources are ranked accordingly. In detail, by using this algorithm PMs are re-ranked according to the selected optimization scheme and based on their VM usage. For example we use as data set resource information from 24 h monitoring and as training set a seven day resource usage monitoring. The analytics are (a) according to utilization levels over time by characterizing it as low, medium and heavy and (b) according to continues data (e.g. memory percent that increases over time). The algorithm performs a weighting process for the selected PMs according to different features (e.g. CPU, RAM percentage).

Step 5. The scheduler sends a request to the database service to record the operation.
Step 6. The scheduler sends a request to the queue of the cloud hosting service with the selected PM for placement.
Step 7. The service sends a request to the cloud hosting service of the selected PM.
Step 8. The selected PM hypervisor launches the VM and the operation is recorded to the database service.

To achieve real time analytics, we implemented a monitoring engine that allows online resource usage monitoring data collection from VMs. The engine is capable of collecting system data based on interval and stores it to an online cloud service that makes it available for data processing. Data is collected every a tiny time interval (e.g. 1 s) and is stored in a temporary local file. The engine includes a number of monitoring features such as CPU (user, nice, system, idle, percent), memory (total, available, percent, used, free, active, inactive, wired), swap (total, used, free, percent, sin, sout), disk usage (total, used, free percent), IO (read count, write count, read bytes, write bytes, read time, write time) according to the *psutil* cross-platform library[2] for collecting information on running processes and system utilization.

3.2. Optimization schemes

The aim of this optimization schemes is to define the weight of the PM according to the resource usage of the VMs. This will reveal information about the already deployed VMs status, like indications that a workload is running or not. To achieve this we provide two optimization schemes. Firstly a classification of the VM status about its current resource usage according to history records (using SVM and SVR) and secondly a convolution method for comparing signals and find their similarity. To achieve this we run well known workloads and we tag them accordingly so the monitoring information contain resource usage and a label about the workload status. Fig. 3 demonstrates this process.

In detail, the VMs are constantly monitored while workloads are running. Reports that are generated by the system are stored into an archive for later usage. Further, we perform statistical evaluation using *opt1* that involves classification and regression using SVM and SVR respectively or *opt2* that involves convolution in the signal to identify similarities and/or patterns. For example, binary classification can allow us to classify a VM according to its resource usage (e.g. to low, medium and high utilization) while the convolution method will allow to find patterns according to workloads stored in an archive. Thus, we first classify VMs and in case a VM has high CPU usage we try to identify if there is a pattern in the CPU usage. If this is the case, the VM is tagged as *running* and we set its weight accordingly. VMs with high CPU usage matched with workload running are assigned a small weight, while VMs with medium CPU usage matched with running workload are assigned with a medium value weight. High value weights are assigned to VMs that have low CPU utilization and no matched workloads. The weight is calculated as *w=CPU_utilization_percent-1*.

We use SVM and SVR methods for binary classification and regression according to resource utilization status. Lets consider the example dataset of CPU (CPU percent, memory percent and disk usage percent) in x,y plane and we represent feature geometrically by vectors. We first find the linear decision hyperplane that separates the features and has the largest

[2] https://pypi.python.org/pypi/psutil.

margin and we train SVM by minimizing the error function given by the formula $\frac{1}{2}w^Tw + C\sum_{1}^{n} a_i$, where C is the capacity constraint, a the parameters for handling no separable data and n are the training data. In addition, to solve for continues data we use the SVR that works as follows. SVR finds a function with at most e-deviation from the target y. The problem can be represented as a convex optimization problem given by the formula $\min\frac{1}{2}\|w\|^2$.

In our study we consider a scenario of a PM p_i, with $p_i \in P$ where P is the total number of servers, and it hosts a VM v_j with $v_j \in V_{p_i}$, where V_{p_i} defines the VMs per server. Each P, V have a resource $R = \begin{bmatrix} c & m & d \end{bmatrix}$ where c is the cpu, m is the memory, s the swap memory, d disk and io the IO usage. We can define the total cluster size

(T) as: $p_{c_T} = \sum_{i=1}^{P} p_c(i)$, $p_{m_T} = \sum_{i=1}^{P} p_m(i)$ and $p_{d_T} = \sum_{i=1}^{P} p_s(i)$.

Similarly, $v_{c_T} = \sum_{i=1}^{V} v_{p_c(i)}$, $v_{m_T} = \sum_{i=1}^{V} v_{p_m(i)}$ and $v_{d_T} = \sum_{i=1}^{V} v_{p_s(i)}$ are defined as the total VM size per server (p).

For each VM placement request, v_ν with $R\nu = [c\nu \quad m\nu \quad d\nu]$ the algorithm follows the next steps. For each $p_i \in P$ it collects the R data and if $p_{c_T} < v_{\nu c_T}$ and $p_{m_T} < v_{\nu m_T}$ and $p_{d_T} < v_{\nu d_T}$ it creates an

$$M = \begin{array}{c} \\ p_1 \\ \dots \\ p_q \end{array} \begin{array}{c} \begin{array}{ccc} c & m & d \end{array} \\ \begin{pmatrix} c_1 & m_1 & d_1 \\ \dots & \dots & \dots \\ c_q & m_q & d_q \end{pmatrix} \end{array}$$

where q is the maximum number of candidate nodes to host. We define a coefficient *coef* as a constraint for sorting the M matrix. If *coef* is set to 1 we sort the matrix according to the cpu size thus the PM p_m with the higher cpu will be the first at the list. It should be mentioned that the default sorting parameter for OpenStack is the memory.

For each p_i, with $p_i \in P$ we follow an optimization scheme to model the selection of PMs. The algorithm can perform predictions on selected criteria based on two properties (a) classification where the output variable takes class labels e.g. for CPU utilization levels (idle, medium and high) and (b) regression where the output variable takes continuous values e.g. for CPU, memory percentages etc.

To do this we use a model, where we have a given dataset $D = (x_1, y_1), (x_2,$
$y_2), \dots, (x_w, y_w)$ where $x_i = (x_{i1}, x_{i2}, \dots, x_{i\nu})$ with x is the input vector and $x =\subseteq \Re^\nu$ and y is the label need to classify and $y_i \in 1, -1$. We build the linear function as follows $f(x) = \sum_{i=1}^{\nu}(w_i \times x_i) + \beta$ where w is the weight vector $w = (w_1, w_2, \dots, w_r)$ and β is bian If $f(x_i) \geq 0$ then the vector $x_i \geq 0$ and y_i

$$y_i = \begin{cases} 1 & \sum_{i=1}^{\nu}(w_i \times x_i) + \beta \geq 0 \\ -1 & \sum_{i=1}^{\nu}(w_i \times x_i) + \beta < 0 \end{cases}$$

SVR uses the same principles as the SVM for classification, by including a margin of tolerance (epsilon) that is set in approximation to the SVM. The algorithm calculates the model accuracies according to the incoming data per interval cl_int that is measured in seconds. The output of the algorithm predicts resource usage (e.g. utilization levels) for the selected features (i.e. cpu percent and memory percentages. We set the algorithm to the following configurations. Having define the SVM and SVR models that are responsible for classifying signals we correlate incoming signals to already classified ones for pattern recognition. For signal comparisons we use cross-correlation. This is defined by scaling a basis function P(Xi), shifted by an offset X, to fit a set of data Di with associated errors σ_i measured at positions Xi as given by the following formula (2).

$$CCF = \frac{\sum_{i=1}^{\nu} P(X_i - X)D_i/\sigma_i^2}{\sum_{i=1}^{\nu} P^2(X_i - X)D_i/\sigma_i^2} \tag{2}$$

To find the similarity between the two signals, we use convolution neural networks. In convolutional neural networks every network layer acts as a detection filter for the presence of specific features or patterns present in the original data. The first layers detect (large) features that can be recognized and interpreted relatively easy. Later layers detect increasingly features that are more abstract (and are usually present in many of the larger features detected by earlier layers). The last layer is able to make an ultra-specific classification by combining all the specific features detected by the previous layers in the input data. In our work, we use convolution neural networks to compare signals and identify their similarity. We extract

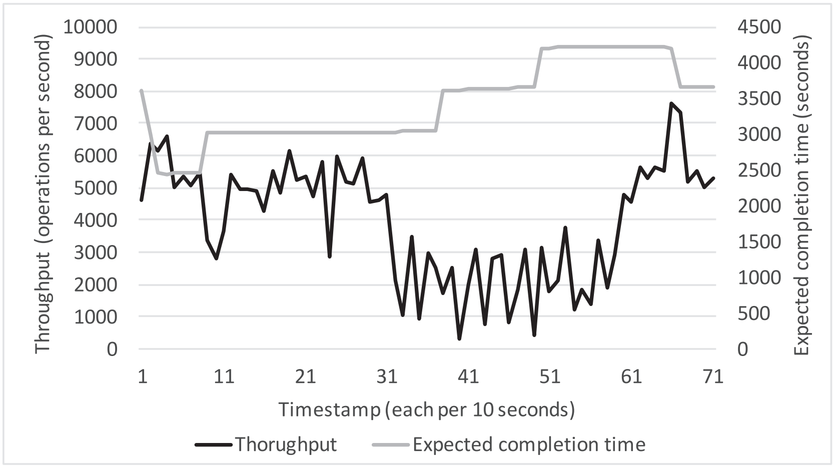

Fig. 4. OpenStack VM placement influence to the Apache Cassandra throughput.

the similarity by empirically identity local maxima in the convolution signal. In particular, we set distance thresholds for identifying maxima and compare between them.

4. Evaluation

This section presents the performance evaluation of the proposed system. This work focuses on the VM scheduling process in Cloud systems. A key problem that Cloud-based schedulers has to address is the dynamic physical host selection that is based not only in static criteria but in a more complete knowledge on what has been executed in the system over time. As discussed before, the hosts weighting strategy is based instant based evaluations, such as current available memory, yet, understanding data to enhance scheduling could be in particular useful and this refers to learning from “both static and dynamic usage statistics”[17]. The motivation of this work is based on two experimental cases, in which we observed performance degradation in the deployed VMs while we schedule and place new VMs in OpenStack as described in 4.1 and in 4.2. The experiments executed in an OpenStack cluster of three PMs in which the controller node has 32GB or RAM and the two compute nodes have 16GB of RAM (rest setup for all nodes includes 16 Cores and 100GB disk). The OpenStack scheduler is the filtering and weighting according to the highest available RAM.

4.1. OpenStack influence in VMs

Firstly, we deploy an Apache Cassandra node in OpenStack and we execute the YCSB workload (100 thousand reads and updates). During the reading phase, we execute a VM scheduling process that places a VM in the controller host (since it has the most available memory at that time) and we observe the resource utilization levels and Cassandra throughput metrics. Fig. 4 demonstrates the variation of the Cassandra throughput and the workload expected completion time while we create a new VM. We run the experiment 10 times and we measure the average for each timestamp (the sampling rate is 10 s). In particular, between timestamps 7 and 13 we create a small size VM of a typical Cirros image and between timestamps 30 and 65 we create a medium size Ubuntu image (2 CPUs, 4GB RAM and 40GB HD) with preinstalled Cassandra. It should be mentioned that the VMs are placed in the controller node that already hosts the Cassandra VM. We can observe that during these two OpenStack processes, the Cassandra throughput drops while the expected estimation time increases. Especially in the second case (of the medium size image) the VM placement penalizes the deployed system performance.

The example shows that in our case a more refined VM placement could be able to enhance the VM scheduling process by “understanding” current VM behaviour (e.g. that placement in this node will drop throughput). An alternative solution would be to select an available PM that will not influence the performance of the already deployed system.

To demonstrate the effectiveness of our solution (called VMA), we run an example case in an OpenStack cluster of 3 PMs (1 controller (32GB RAM) and 2 compute nodes (16GB RAM)). We first run the default OpenStack scheduler in order to place 3 VMs of a pre-installed Cassandra cluster that are placed in PM_1, PM_2 and PM_3 respectively. Then we execute a YCSB workload to PM_1 while PM_2 and PM_3 are in idle state and we execute one more VM placement. We observed that the default VM scheduler forces the VM to be placed in PM_1 (which at that time instance has the most available RAM), however our scheduler selects PM_2 due to the fact that recognizes the increased resource usage of PM_1.

Fig. 5 demonstrates the throughput in both cases. In particular the grey line demonstrates the throughput in the default case (where the VM is placed in the same host) while the black line is based on our scheduler. We can observe that our solution does not affect the throughput (that in the default case drops almost to 0 between timestamps 17 and 19). For this specific scenario (of 100.000 YCSB reads and updates) our solution optimizes throughput averagely by 19%.

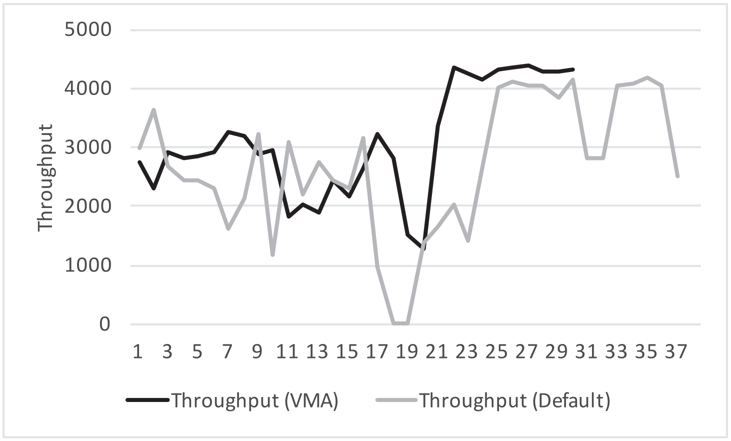

Fig. 5. Prediction of MongoDB CPU percent when running YCSB.

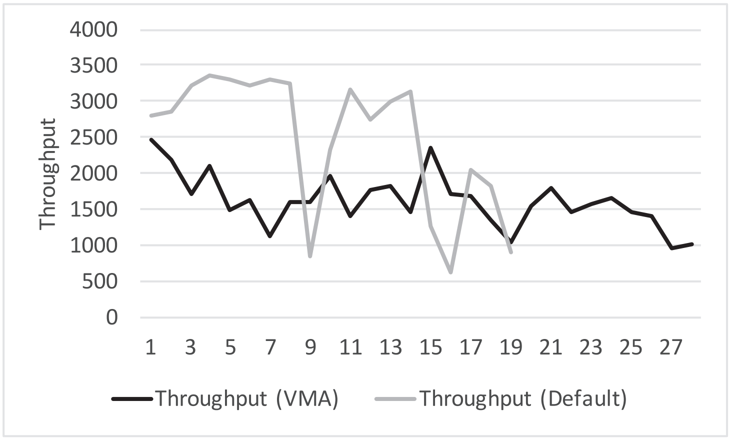

Fig. 6. Prediction of Elasticsearch VM memory percent when running YCSB.

In the second case, we execute a similar experiment for the MongoDB system. As before, we deploy 3 MongoDB instances in OpenStack and we run a YCSB workload (of 100.000 YCSB reads and updates) in the PM_1 while PM_2 and PM_3 are in idle state. Fig. 6 demonstrates the throughput comparison between our solution and the default OpenStack scheduler. After, we create a new VM and the default OpenStack scheduler places it in the PM_1, thus affecting the performance of the running workload. We can observe that even if the signal is not so stable (as in the Cassandra case), in average the OpenStack placement process affects the throughput (represented by the black line). We calculated that averagely our solution offers 26% performance optimization over the default scheduler.

Finally, Fig. 7 demonstrates the Elasticsearch performance measurements that include (a) total runtime calculation and (b) total throughput.

Similarly to the two previous experiments we run 3 instances over OpenStack and we compare default versus VMA solution. In detail, VMA solution offers 13% optimization over the default scheduler. To conclude, these experiments demonstrate that default OpenStack scheduling affects already deployed systems performance. Our solution refines scheduling by evaluating VM resource usage, thus VM placements are happening accordingly (for example in PMs that contain under-utilized VMs).

4.2. VMs CPU steal time

In this experiment we observe the CPU steal time (that reflects the time that a hypervisor services another virtual processor) in two Cassandra nodes. Fig. 8 demonstrates the CPU steal time between a single VM and two VMs that are executed concurrently in an OpenStack cloud system. We used the real world workload of YCSB and we executed 10 thousand reads and updates respectively in both Cassandra nodes. We executed the same workload for both VMs and we ran 10 experiments and we calculated the average of the overall tests. We conclude to the following observations. When we increase the VMs from one to two the CPU steal time is increasing. For example for a single VM the CPU steal time is around 6% while

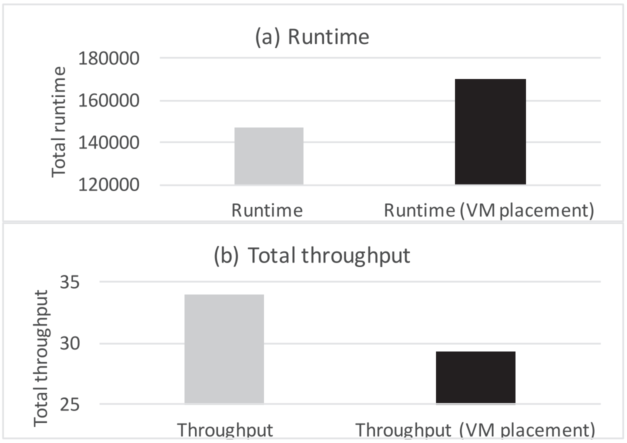

Fig. 7. Prediction of Elasticsearch CPU percent when running YCSB.

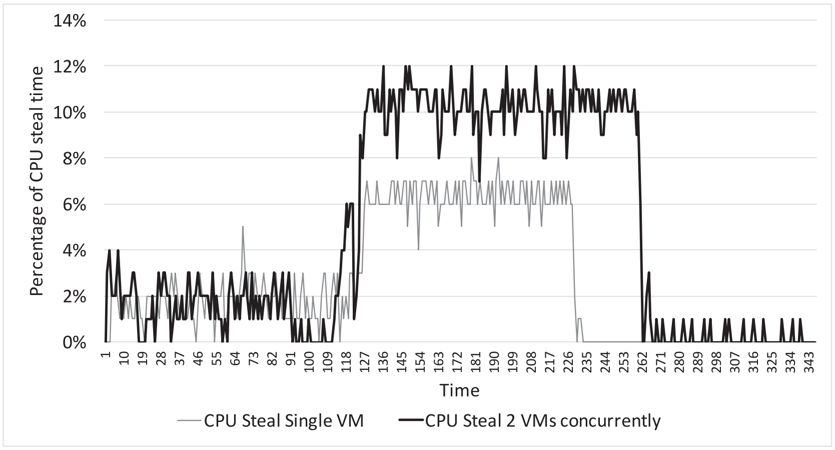

Fig. 8. Steal time percentage for YCSB Apache Cassandra workload execution in small size VM for 10.000 "inserts" and "updates" in an OpenStack system.

when we launch a second VM in the same PM, the CPU steal time is increased to 10%. Based on this simple experiment we can conclude that by increasing the number of VMs the percentage of the CPU steal time is also increased and this affects VM utilization.

To calculate the actual resource usage, we introduce the factor of the "real" CPU utilization that is related to the amount of CPU that subtracts the amount that has been "stolen" (or in our case has not be offered by the hypervisor to the running VM). Fig. 9 demonstrates the average value of the "real" CPU utilization rates for the YCSB read and update phases from 10 runs. We observe (for the YCSB updating phase between the timestamps 220 and 365) that while the system CPU utilization is 81%, when we launch the seconds VM this value drops to 73%, so resource utilization in reality it is dropped by 8%.

Another experimental case involves execution of the YCSB Apache Cassandra workload in a medium size VM that has been deployed in Amazon EC2. In particular, we run 50.000 inserts and 50.000 updates and we observe the CPU steal time. The time series in "x" axis represent the time, while in "y" axis the CPU steal time over the workload execution (its time point represent the measurement of the steal time in relation to the previous point, for example from 6.88 to 6.89 represents CPU steal time of 1%). Fig. 10 demonstrates that during 10 min, the CPU steal time percentage was overall 10% (increased from 6.88 to 6.98). Based on this discussion we conclude that CPU steal time is an important factor to take in mind during VM scheduling as it can significantly affects VMs CPU utilization levels. A more refined VM scheduling can be based on predicting the CPU steal time according to the real time resource usage in order to perform scheduling that minimizes the CPU steal time.

To demonstrate our solution we run a CPU steal case in an OpenStack cluster of 2 PMs (PM_1 and PM_2). In our case, PM_1 has 32GB or RAM while PM_2 has 16GB of RAM. After, we place one VM to PM_1 and we run the YCSB workload respectively. Fig. 11 shows the statistical CPU steal time distribution in an x,y plane. We can observe that the CPU steal time (default) is higher than our solution (that minimizes the overall steal time).

We run a similar experiment in a MongoDB system using again two PMs. As previous case, the default OpenStack scheduler selects places the new VM to PM_1 that has already the MongoDB system. Fig. 12 demonstrates the comparison of the

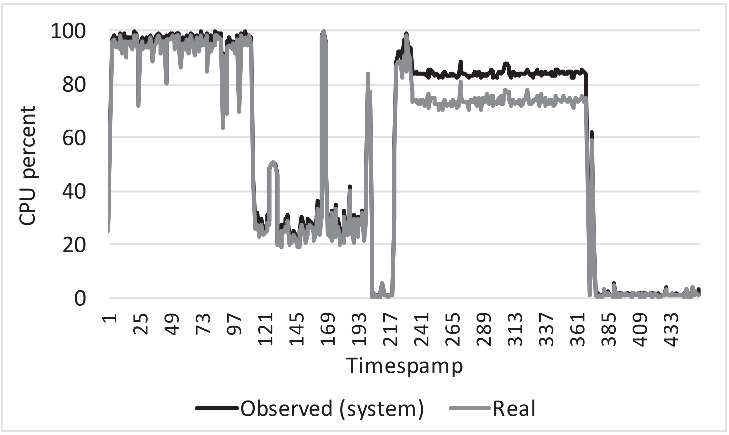

Fig. 9. "Real" CPU utilization percentage for YCSB Apache Cassandra workload execution in small size VM for 10.000 "inserts" and 10.000 "updates" records in OpenStack.

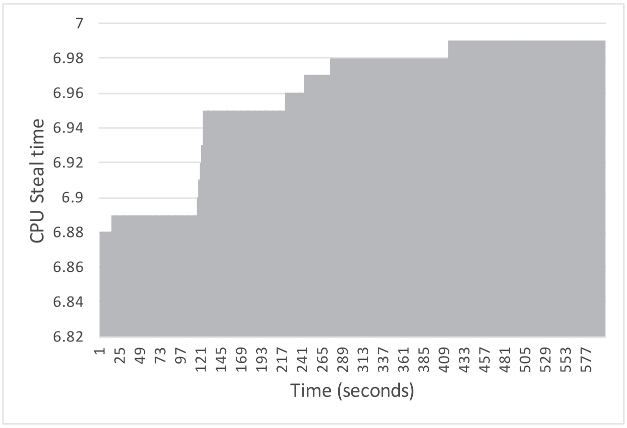

Fig. 10. "CPU steal time" for YCSB Apache Cassandra workload execution in medium size VM for 50.000 "inserts" and 50.000 "updates" records in an Amazon EC2 instance.

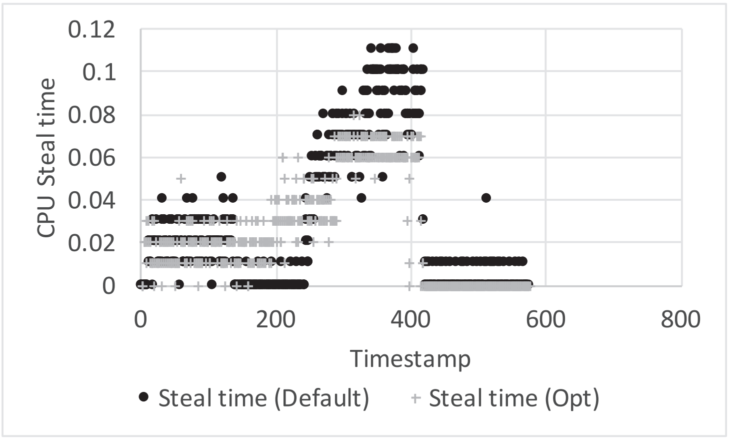

Fig. 11. Comparison of default OpenStack scheduler versus VMA for CPU steal time when running YCSB in Cassandra node.

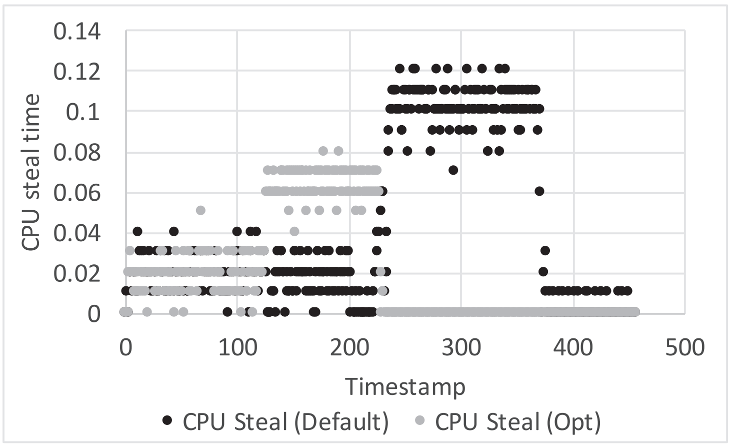

Fig. 12. Comparison of default OpenStack scheduler versus VMA for CPU steal time in MongoDB.

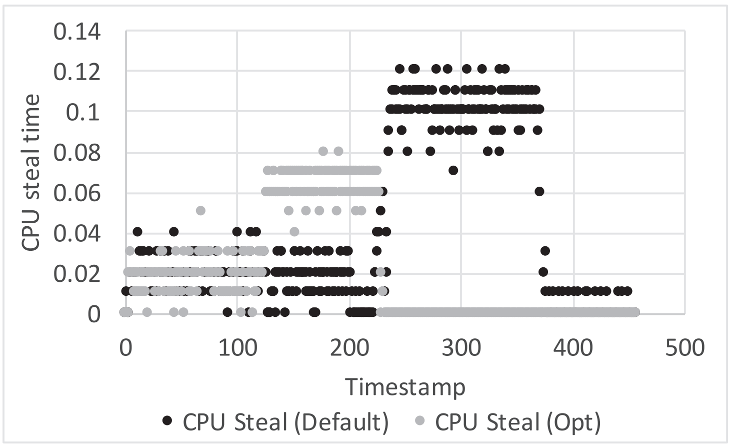

Fig. 13. Comparison of default OpenStack scheduler versus VMA for CPU steal time in Elasticsearch node.

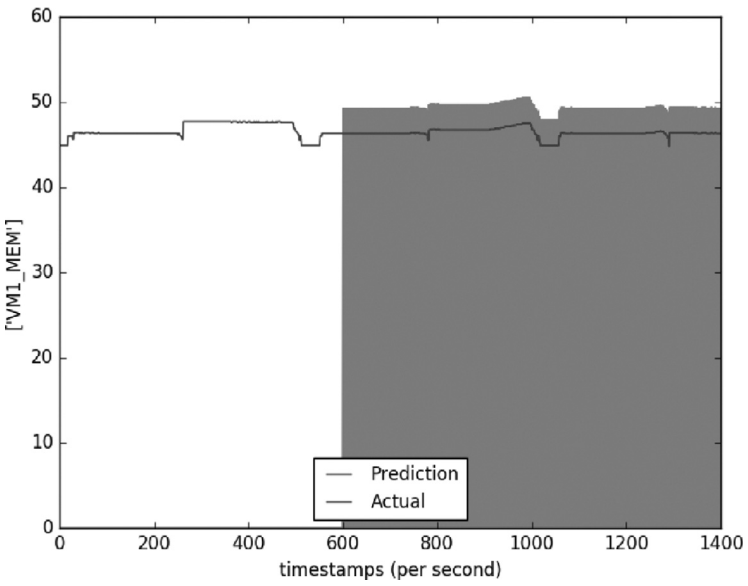

Fig. 14. Prediction of Cassandra VM memory percent when running YCSB.

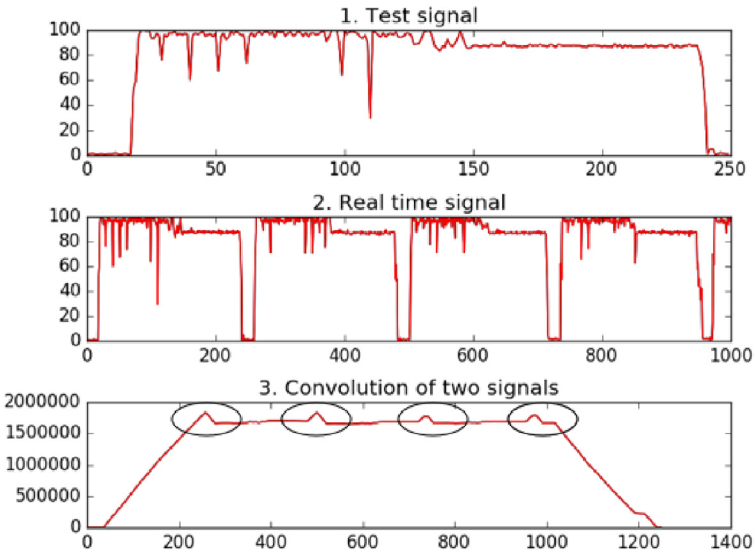

Fig. 15. Prediction of MongoDB CPU percent when running YCSB.

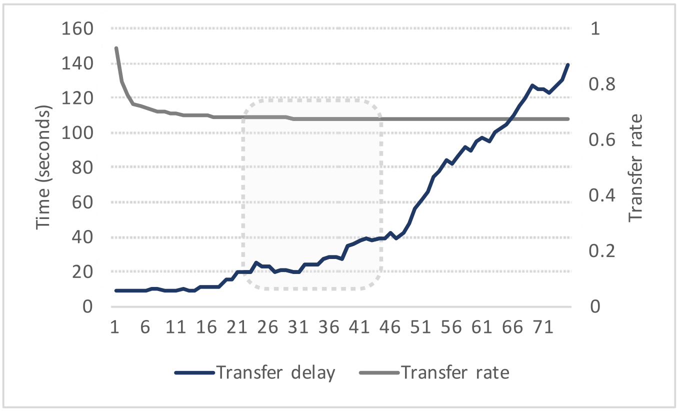

Fig. 16. Real time data transfer rates.

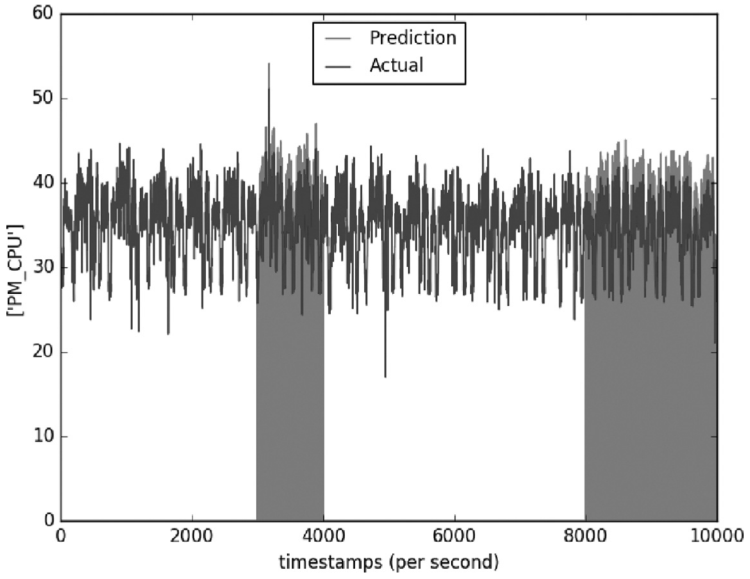

Fig. 17. Prediction of Cassandra CPU percent when running YCSB.

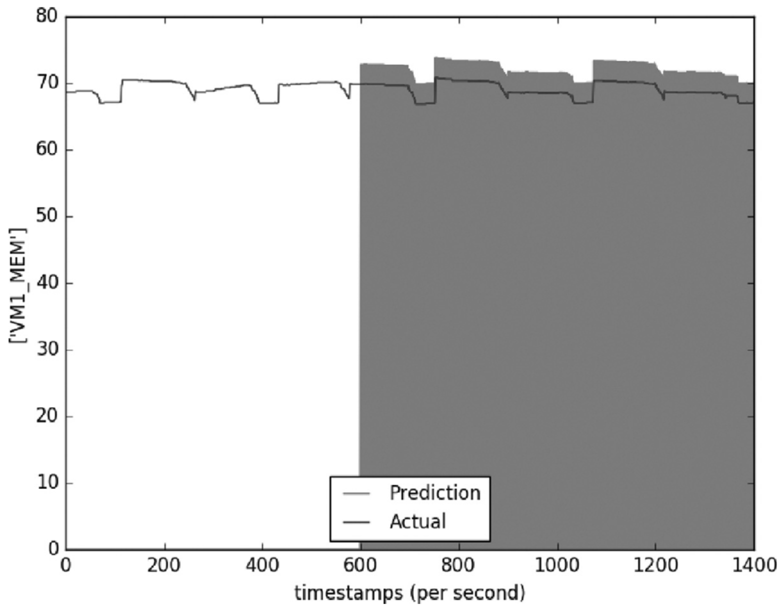

Fig. 18. Prediction of MongoDB VM memory percent when running YCSB.

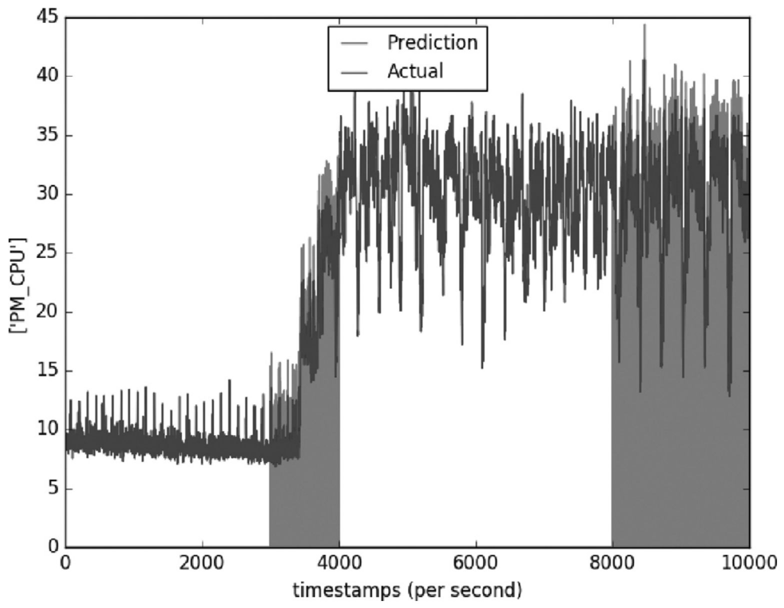

Fig. 19. Prediction of MongoDB CPU percent when running YCSB.

CPU steal time for both cases. We can observe that our solution outperforms the default one. In detail, the average CPU steal time for the default scheduling case is 4% while for the VMA scheduler we reduce this value to 2%.

Finally, we run an experiment by deploying an Elasticsearch system as shown in Fig. 13.

Our experiment demonstrates the comparison of default OpenStack scheduler versus VMA for CPU steal time when running YCSB in Elasticsearch node. Similarly to previous experiments VMA outperforms default scheduler.

4.3. Analysis of the tools

This section presents the analysis of the tools and their performance. We include (a) the results related with the accuracy of the prediction models (that use SVR), (b) the results related with the signal similarity comparison and (c) performance measures of the real time resource monitoring system.

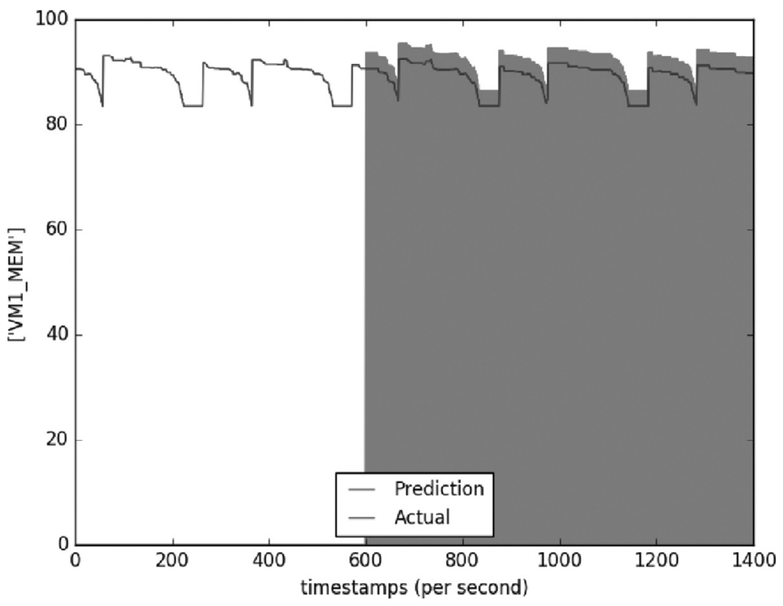

Fig. 20. Prediction of Elasticsearch VM memory percent when running YCSB.

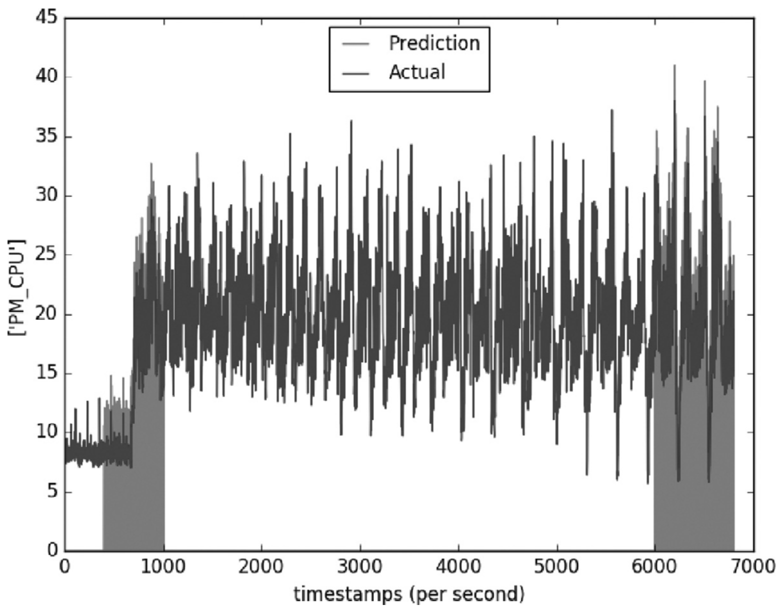

Fig. 21. Prediction of Elasticsearch CPU percent when running YCSB.

4.3.1. Analysis of the predictions

To demonstrate the accuracy of the prediction model we train our model with data from 24 h by running YCSB workloads. Then we select a window of real time data collected by the system and the actual prediction. To demonstrate the patterns in a more clear manner we increase the prediction rates by 5%. Fig. 14 demonstrates the Cassandra VM memory usage by running a typical YCSB workload (that includes 100 thousands reads and writes). Rest figures in Appendix show the prediction analysis for the rest of the deployed systems (including MongoDB and Elasticsearch).

In particular, Fig. 17 (page 37)demonstrates the CPU utilization levels and the prediction during two different timeframes. Figs. 18 (page 38) and 19 (page 38) demonstrate the accuracy of our algorithm for predicting the memory and CPU resource usage of the MongoDB system respectively. Finally, Figs. 20 and 21 (page 39) demonstrate the accuracy of our algorithm for predicting the memory and CPU resource usage of the Elasticsearch system respectively. For all cases, it is apparent that the prediction is similar to the real usage pattern.

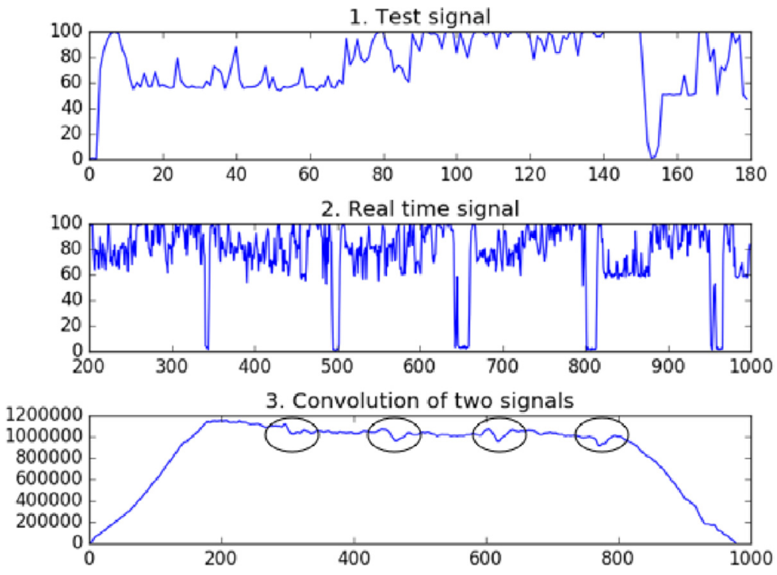

Fig. 22. Prediction of Elasticsearch VM memory percent when running YCSB.

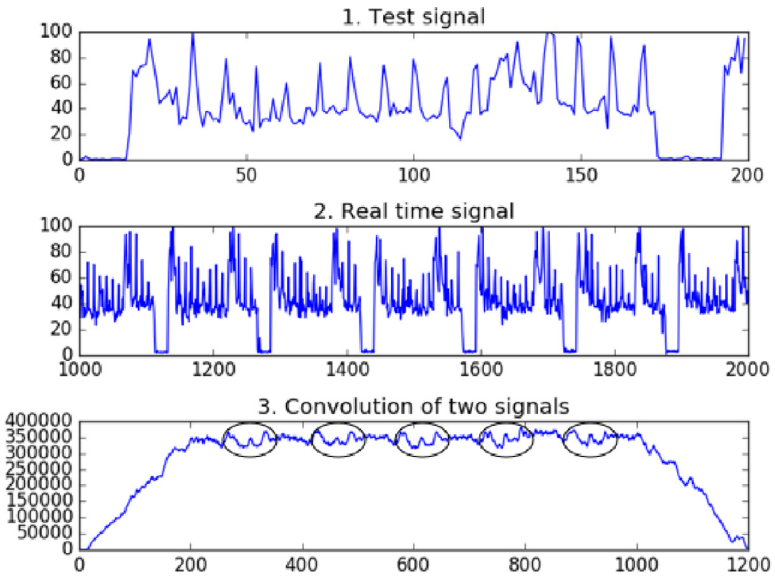

Fig. 23. Prediction of Elasticsearch CPU percent when running YCSB.

Based on this analysis, we can conclude that for both CPU and memory, the workload executions demonstrate a repeating pattern. Thus, in our case we evaluate the workload executions (e.g. the last hour) by feeding this records to the model, and we identify if a VM is in the middle of a workload execution or not by comparing sequences of signals. In case of signal matching we classify the VM as "running" or not.

4.3.2. Analysis of signal similarity using convolution

In this section we present the analysis of the signal similarity algorithm.

We used the convolution method in order to compare the signals and we set a comparison coefficiency that is related with the similarity of the convolution comparison. Fig. 15 demonstrates the signal similarity and the pattern extraction between the test signal and the real time signal. It should be mentioned that test signals are stored into the archive, so the more the samples, higher the accuracy of our model. We present the similarity comparison and the convolution signal in the Appendix section (MongoDB signal is in page 40 and Elasticsearch in page 40). We can observe that when there are 4 repetitions between the signals meaning that a workload is running. In the specific signal, we can observe that the local maximum points demonstrate the peaks of the pattern matching.

4.3.3. *Real time monitoring engine benchmark analysis*

In this section we present the analysis of the real time monitoring engine that is preinstalled and configured to each We set the data collection interval to 1 s and we aim to explore what is the timeframe in which data should be recorded and forwarded to the cloud. The machine learning engine connects to a cloud service and parses the data in an iteratively mode. It executes a series of machine learning models in order to calculate the best accuracy score, thus to conclude with the best model for VM placement. The data is stores in csv format and are the input to the machine learning model. Fig. 16 demonstrates the transfer delay versus the rate (that is given by the equation $rate = \frac{rows-count}{rows}$ where rows are the uploaded rows per iteration and count is the total rows stored in an array).

We observe that the transfer delay is increased significantly after the *timestamp 50* where each timestamp is a window of 1 s. We can also notice that the transfer rate keeps a steady trend line at that time window. Based on these facts we set the transfer point in 50 s, and we perform the iterations after every successful completion of a machine learning round. It should be mentioned that each learning round includes the measurement of accuracies for selected models.

Since data are uploaded dynamically, to achieve up to modelling of real time data we force the engine to load data for each iteration. This means that the system will spend a significant amount of time to load the data, thus the actual modelling is happening based on this delay. For example data upload rate is 3,37 MB per second meaning that in this example case a file size of 2.84 MB will be uploaded in 9.56 s. Thus the machine learning model refers to data collected before that time.

5. Conclusions

Cloud systems use different virtual machine (VM) placement algorithms to schedule instances by selecting physical machines (PMs) according to real time system information (i.e usage of CPU, memory, hard disk, network and other). In this work we stress the problem of instantaneously collected system data that in many cases does not reflect the big picture (i.e. average resource utilization levels). The current VM placement does not consider real time VM resource utilization levels. In this work we propose a new VM placement algorithm based on past VM usage experiences. We monitor the VM usage in real time and we train different machine learning models to calculate the prediction of the VM resource usage per server, thus to place VMs accordingly. We present an algorithm that allows self managed VM placement according to PM and VM utilization levels. Usually, traditional systems (i.e. OpenStack) use a filtering (which PMs can host the VM thus having resources) and weighing method (which PM has the higher RAM) to select PMs *based on the specific time instance*, without considering the actual VMs' resource usage of the selected PMs. We introduce the concept of analyzing past VM resource usage according to historical records based on computational learning to optimize the PM selection phase.

Also, we proposed a self managed VM placement algorithm based on real time virtual resource monitoring that utilizes machine learning models to train and learn from past virtual resources usage. Thus, we developed a monitoring engine that collects resource usage data on real time. The VMA collects data and applies different models for class labels (classification) and/or continues values (regression). The proposed algorithm allows data processing based on a timeframe window to define the actual behaviour of the PMs or VMs. The experimental analysis is prosperous and results highlight major improvements in the VM placement process. The future steps of our research include further experimental analysis and experimentation related to different machine learning models.

References

[1] A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, Concurr. Comput. 24 (13) (2012) 1397–1420, doi:10.1002/cpe.1867.

[2] N. Bessis, S. Sotiriadis, V. Cristea, F. Pop, Modelling requirements for enabling meta-scheduling in inter-clouds and inter-enterprises, in: Proceedings of the 2011 Third International Conference on Intelligent Networking and Collaborative Systems, in: INCOS '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 149–156, doi:10.1109/INCoS.2011.120.

[3] E. Bin, O. Biran, O. Boni, E. Hadad, E.K. Kolodner, Y. Moatti, D.H. Lorenz, Guaranteeing high availability goals for virtual machine placement, in: Distributed Computing Systems (ICDCS), 2011 31st International Conference on, 2011, pp. 700–709, doi:10.1109/ICDCS.2011.72.

[4] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, E. Silvera, A stable network-aware VM placement for cloud systems, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012), in: CCGRID '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 498–506, doi:10.1109/CCGrid.2012.119.

[5] M. Cardosa, M.R. Korupolu, A. Singh, Shares and utilities based power consolidation in virtualized server environments, in: Proceedings of the 11th IFIP/IEEE International Conference on Symposium on Integrated Network Management, in: IM'09, IEEE Press, Piscataway, NJ, USA, 2009, pp. 327–334. URL http://dl.acm.org/citation.cfm?id=1688933.1688986.

[6] S. Chaisiri, B.S. Lee, D. Niyato, Optimization of resource provisioning cost in cloud computing, IEEE Trans. Serv. Comput. 5 (2) (2012) 164–177, doi:10.1109/TSC.2011.7.

[7] A. Corradi, M. Fanelli, L. Foschini, Vm consolidation: a real case based on openstack cloud, Future Gener. Comput. Syst. 32 (2014) 118–127, doi:10.1016/j.future.2012.05.012.

[8] A.V. Do, J. Chen, C. Wang, Y.C. Lee, A.Y. Zomaya, B.B. Zhou, Profiling applications for virtual machine placement in clouds, in: Cloud Computing (CLOUD), 2011 IEEE International Conference on, 2011, pp. 660–667, doi:10.1109/CLOUD.2011.75.

[9] C. Dupont, T. Schulze, G. Giuliani, A. Somov, F. Hermenier, An energy aware framework for virtual machine placement in cloud federated data centres, in: Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet, in: e-Energy '12, ACM, New York, NY, USA, 2012, pp. 4:1–4:10, doi:10.1145/2208828.2208832.

[10] E. Elmroth, L. Larsson, Interfaces for placement, migration, and monitoring of virtual machines in federated clouds, in: Grid and Cooperative Computing, 2009. GCC '09. 8th International Conference on, 2009, pp. 253–260, doi:10.1109/GCC.2009.36.

[11] W. Fang, X. Liang, S. Li, L. Chiaraviglio, N. Xiong, Vmplanner: optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers, Comput. Netw. 57 (1) (2013) 179–196. http://dx.doi.org/10.1016/j.comnet.2012.09.008.

[12] T. Fitfield, Introduction to openstack, Linux J. 2013 (235) (2013). URL: http://dl.acm.org/citation.cfm?id=2555789.2555793
[13] R. Folco, Openstack nova scheduler: disable RAM weigher, 2015.
[14] Y. Gao, H. Guan, Z. Qi, Y. Hou, L. Liu, A multi-objective ant colony system algorithm for virtual machine placement in cloud computing, J. Comput. Syst. Sci. 79 (8) (2013) 1230–1242, doi:10.1016/j.jcss.2013.02.004.
[15] H. Goudarzi, M. Ghasemazar, M. Pedram, Sla-based optimization of power and migration cost in cloud computing, in: Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, 2012, pp. 172–179, doi:10.1109/CCGrid.2012.112.
[16] H. Goudarzi, M. Pedram, Energy-efficient virtual machine replication and placement in a cloud computing system, in: Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, in: CLOUD '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 750–757, doi:10.1109/CLOUD.2012.107.
[17] Intel-opensource.org, Utilization based scheduling in openstack compute (nova), 2015.
[18] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, E. Snible, Improving performance and availability of services hosted on IaaS clouds with structural constraint-aware virtual machine placement, in: Services Computing (SCC), 2011 IEEE International Conference on, 2011, pp. 72–79, doi:10.1109/SCC.2011.28.
[19] J. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang, Joint VM placement and routing for data center traffic engineering, in: INFOCOM, 2012 Proceedings IEEE, 2012, pp. 2876–2880, doi:10.1109/INFCOM.2012.6195719.
[20] O. Khedher, Mastering OpenStack, 978-1-78439-564-3, Packt Publishing, July 2015.
[21] G. Kousiouris, T. Cucinotta, T. Varvarigou, The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks, J. Syst. Softw. 84 (8) (2011) 1270–1291, doi:10.1016/j.jss.2011.04.013.
[22] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, T.D. Nguyen, Reducing electricity cost through virtual machine placement in high performance computing clouds, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, in: SC '11, ACM, New York, NY, USA, 2011, pp. 22:1–22:12, doi:10.1145/2063384.2063413.
[23] B. Li, J. Li, J. Huai, T. Wo, Q. Li, L. Zhong, Enacloud: an energy-saving application live placement approach for cloud computing environments, in: Cloud Computing, 2009. CLOUD '09. IEEE International Conference on, 2009, pp. 17–24, doi:10.1109/CLOUD.2009.72.
[24] W. Lloyd, S. Pallickara, O. David, M. Arabi, K. Rojas, Dynamic scaling for service oriented applications: implications of virtual machine placement on IaaS clouds, in: Cloud Engineering (IC2E), 2014 IEEE International Conference on, 2014, pp. 271–276, doi:10.1109/IC2E.2014.40.
[25] J.L. Lucas-Simarro, R. Moreno-Vozmediano, R.S. Montero, I.M. Llorente, Scheduling strategies for optimal service deployment across multiple clouds, Future Gener. Comput. Syst. 29 (6) (2013) 1431–1441, doi:10.1016/j.future.2012.01.007.
[26] M. Marzolla, O. Babaoglu, F. Panzieri, Server consolidation in clouds through gossiping, in: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a, 2011, pp. 1–6, doi:10.1109/WoWMoM.2011.5986483.
[27] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: Proceedings of the 29th Conference on Information Communications, in: INFOCOM'10, IEEE Press, Piscataway, NJ, USA, 2010, pp. 1154–1162. URL: http://dl.acm.org/citation.cfm?id=1833515.1833690
[28] K. Mills, J. Filliben, C. Dabrowski, Comparing vm-placement algorithms for on-demand clouds, in: Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, in: CLOUDCOM '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 91–98, doi:10.1109/CloudCom.2011.22.
[29] OpenStack, Scheduling, 2016.
[30] J.T. Piao, J. Yan, A network-aware virtual machine placement and migration approach in cloud computing, in: Proceedings of the 2010 Ninth International Conference on Grid and Cloud Computing, in: GCC '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 87–92, doi:10.1109/GCC.2010.29.
[31] C.-M. Pintea, C. Pascan, M. Hajdu-Macelaru, Comparing several heuristics for a packing problem, CoRR abs/1210.4502 (2012).
[32] S. Sotiriadis, N. Bessis, An inter-cloud bridge system for heterogeneous cloud platforms, Future Gener. Comput. Syst. 45 (2016) 180–194.
[33] S. Sotiriadis, N. Bessis, C. Amza, R. Buyya, Vertical and horizontal elasticity for dynamic virtual machine reconfiguration, IEEE Trans. Serv. Comput. PP (99) (2016), doi:10.1109/TSC.2016.2634024. 1–1
[34] S. Sotiriadis, N. Bessis, A. Anjum, R. Buyya, An inter-cloud meta-scheduling (icms) simulation framework: architecture and evaluation, IEEE Trans. Serv. Comput. PP (99) (2015), doi:10.1109/TSC.2015.2399312. 1–1
[35] S. Sotiriadis, N. Bessis, N. Antonopoulos, Towards inter-cloud schedulers: a survey of meta-scheduling approaches, in: Proceedings of the 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, in: 3PGCIC '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 59–66, doi:10.1109/3PGCIC.2011.19.
[36] S. Sotiriadis, N. Bessis, N. Antonopoulos, A. Anjum, Simic: sesigning a new inter-cloud simulation platform for integrating large-scale resource management, in: 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), 2013, pp. 90–97, doi:10.1109/AINA.2013.123.
[37] S. Sotiriadis, N. Bessis, P. Kuonen, N. Antonopoulos, The inter-cloud meta-scheduling (ICMS) framework, in: Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications, in: AINA '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 64–73, doi:10.1109/AINA.2013.122.
[38] J. Tordsson, R.S. Montero, R. Moreno-Vozmediano, I.M. Llorente, Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers, Future Gener. Comput. Syst. 28 (2) (2012) 358–367, doi:10.1016/j.future.2011.07.003.
[39] F.-H. Tseng, C.-Y. Chen, L.-D. Chou, H.-C. Chao, J.-W. Niu, Service-oriented virtual machine placement optimization for green data center, Mob. Netw. Appl. 20 (5) (2015) 556–566, doi:10.1007/s11036-015-0600-9.
[40] H.N. Van, F.D. Tran, J.M. Menaud, Sla-aware virtual resource management for cloud infrastructures, in: Computer and Information Technology, 2009. CIT '09. 9th IEEE International Conference on, 1, 2009, pp. 357–362, doi:10.1109/CIT.2009.109.
[41] H.N. Van, F.D. Tran, J.M. Menaud, Performance and power management for cloud infrastructures, in: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, 2010, pp. 329–336, doi:10.1109/CLOUD.2010.25.
[42] S. Xi, C. Li, C. Lu, C.D. Gill, M. Xu, L.T.X. Phan, I. Lee, O. Sokolsky, Rt-open stack: CPU resource management for real-time cloud computing, in: Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on, 2015, pp. 179–186, doi:10.1109/CLOUD.2015.33.
[43] J. Xu, J.A.B. Fortes, Multi-objective virtual machine placement in virtualized data center environments, in: Proceedings of the 2010 IEEE/ACM InternationalL Conference on Green Computing and Communications & InternationalL Conference on Cyber, Physical and Social Computing, in: GREENCOM-CPSCOM '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 179–188, doi:10.1109/GreenCom-CPSCom.2010.137.