Self managed virtual machine scheduling in cloud systems

Stelios Sotiriadis

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

e-mail: s.sotiriadis@utoronto.ca

Nik Bessis

Department of Computing, Edge Hill University, St Helens Rd, Ormskirk, Lancashire L39 4QP, UK

e-mail: Nik.Bessis@edgehill.ac.uk

Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Parkville VIC 3010, Australia

e-mail: rbuyya@unimelb.edu.au

Abstract

In today Cloud systems, Virtual Machines (VMs) are scheduled to pyhsical machines according to instant-based resource utilization (e.g. in hosts with most available RAM). However, this does not take into account the overall resource utilization over time, or the affection of scheduling and placement processes that in many cases is computational expensive, into the already deployed systems. To achieve these, we present a Cloud scheduling algorithm based on VM resource utilization prediction using machine learning. We analyze past VM usage in order to schedule VMs in a way that optimizes performance based on two scenarios. Firstly, we observed that Cloud management processes, like VM placement, affect already deployed systems so we aim to minimize such performance degradation. In the experimental study we notice that overloaded VMs tend to steal CPU times from neighbouring VMs, so we aim to increase VMs real CPU utilization. The proposed algorithm is based on real time resource monitoring and refines traditional instant-based physical machine selection VM schedulers. The experimental

analysis compares our solution with traditional schedulers used in OpenStack by exploring the behaviour of NoSQL systems.

Keywords: Cloud computing, OpenStack, virtual machine placement, virtual machine scheduling

---

1. Introduction

Lately, Cloud computing has been emerged as one of the most widely used systems for provisioning virtual resources to everyday users. Many cloud platforms host services (including fundamental services such as infrastructure, platform and software) that are available on a pay as you go model. One of the most important process in such system is the virtual machine (VM) scheduling that determines in which physical machine (PM) resources should be allocated in order to launch a VM instance (a process called VM placement). Simple solutions include VM scheduling to a PM that has available computational resources, is within an availability zone (location constraints set by users) and is ranked as the best available host among others. There are various criteria for VM placement including scheduling to hosts with fewer running instances, according to filtering with regards to current resource utilization levels or multi sub-scheduling layers that allow more refined placement according to network traffic, availability zones etc. In general, a second level of scheduling could provide flexibility over the regular scheduling decision Fitfield (2013).

In this work we propose the concept of online VM scheduling according to resource monitoring data extracted from past utilizations (either PMs or VMs). Our solution, that runs at real time, overcomes traditional solutions in which scheduling is based on selecting the most optimized PM without taking a consideration of the utilization levels of the already deployed VMs. Current systems do not support dynamic VM placement, for example OpenStack performs a filtering and weighing of PMs in a static way, and does not take into consideration past system behaviour. 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* Intel-opensource.org (2015), 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.

We focus on the OpenStack platform[1], that is a well know open source software to build private and public clouds. OpenStack places VMs by selecting the largest memory node until the VMs number exceeds the limit. Such behaviour overloads powerful PMs in the stack and leaves low RAM PMs under-utilized Folco (2015). Similarly in Intel-opensource.org (2015), 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, we can conclude that current OpenStack scheduling is not well refined since only sorts and weighs PMs in a static manner based on instantaneously collected resource usage.

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 and provide indications and predictions as vital factors for VM placement. This is feed as the input in the VM scheduling process. We expect that such solution will provide deeper understanding of system parameters and their affection to the whole system. We also anticipate that it will be the starting point for conceptualizing the behaviour of the system based on computational learning to optimize the VM scheduling phase.

Based on this discussion, the contributions of this work is the proposition of an online VM placement algorithm that is dynamic and adaptive and it is based on historical resource usage of PMs and VMs. We differentiate from the literature approaches as we focus on equating PM and VM resource usage based on pattern extraction and according to a continuously monitoring and data analysis using machine learning. In addition, we explore the *CPU steal time* metric that defines the percentage of the time of a VM CPU waiting for a PM CPU to allocate resources. In other words the hypervisor assigns CPU cycles to other processes (including other "noisy" VM neighbours), thus leading to performance degradation of the VM.

Having said that, Section 2 presents the motivation of this work and Section3 presents an extensive literature review analysis for algorithm and approaches based on cloud VM placement and scheduling. Section 4.2 presents the VM placement algorithm based on the machine learning model. Section 5 presents the description of the OpenStack system, the workload analysis

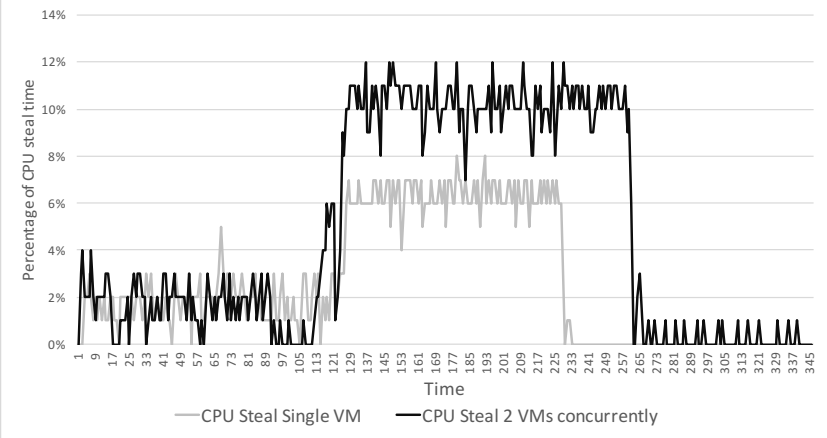
[1]OpenStack: https://www.openstack.org

and the performance evaluation. Finally, Section **??** discusses the conclusions and future research directions.

2. Motivation experiment

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"Intel-opensource.org (2015). Figure 1 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 Yahoo! Cloud Serving Benchmark (YCSB) and we executed 10.000 read and update records respectively in an Apache Cassandra system.

Figure 1: Steal time percentage for YCSB Apache Cassandra workload execution in small size VM for 10.000 records "inserts" and "updates" in an OpenStack system

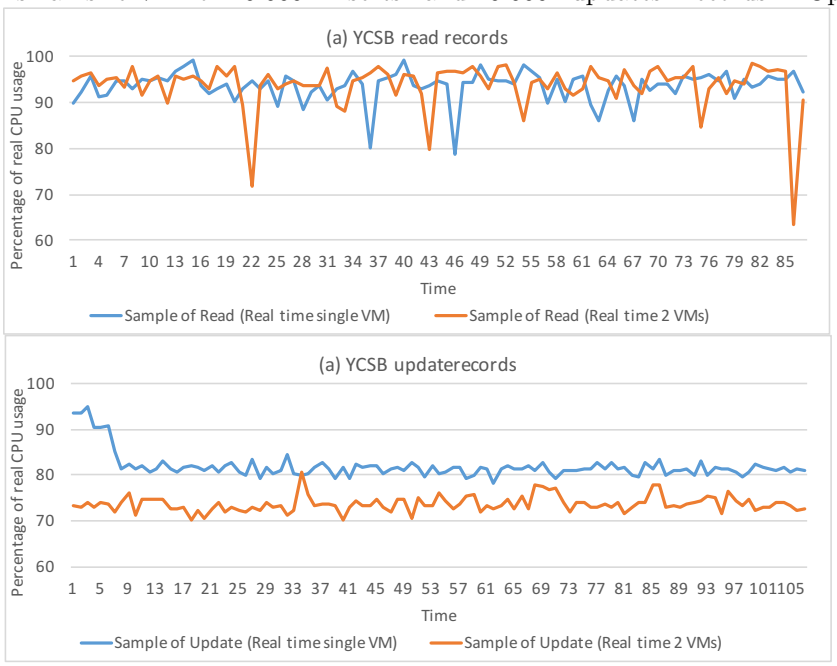


We executed the same workload for both VMs and we conclude to the following observations. When we increased the number of VMs from 1 to 2 (that are executed concurrently in the same PM) the CPU steal time is increasing. For example for a single VM the CPU steal time is around 6% (we ran 10 experiments and we calculated the average of the overall tests)

while when we launch a second VM the CPU steal time is increased to 10% (for the same VM). 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 resource utilization.

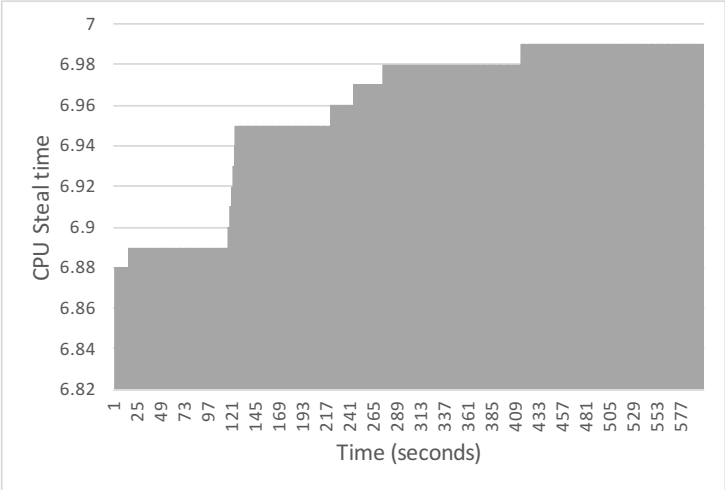
Another important factor is the "real" CPU utilization that is related with the amount of CPU that is used by subtracting the amount that has been stolen (or in our case has not be offered by the hypervisor to the running VM). Figure 2 demonstrates the average value of the "real" CPU utilization rates for the YCSB read and update phases from 10 test cases. In particular, for Figure 2 (a), the real CPU utilization is almost the same (98%), however for the 2 VMs sample, it shows few spikes that represent resource usage degradation (for example at time instance 22). In Figure 2 (b), we can observe that for the case of 2 VMs running concurrently, the "real" CPU usage is significantly decreased. In other words, while the system claims that offers an average of 81% "real" CPU utilization, when we launch the seconds VM this value drops to 73%, so resource utilization in reality is dropped by 8%.

Figure 2: "Real" CPU utlization percentage for YCSB Apache Cassandra workload execution in small size VM for 10.000 "inserts" and 10.000 "updates" records in OpenStack



Another experimental case involves execution of the YCSB Apache Cassandra workload in a medium size VM (2 CPUs, 4 GB RAM and 40 GB Hd) that has been deployed in Amazon EC2. In particular, we run 50.000 insert records and 50.000 update records and we observe the CPU steal time during a 10 minutes window. 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%). Figure 3 demonstrates that during 10 minutes, the CPU steal time percentage was overal 10% (increased from 6.88 to 6.98).

Figure 3: "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



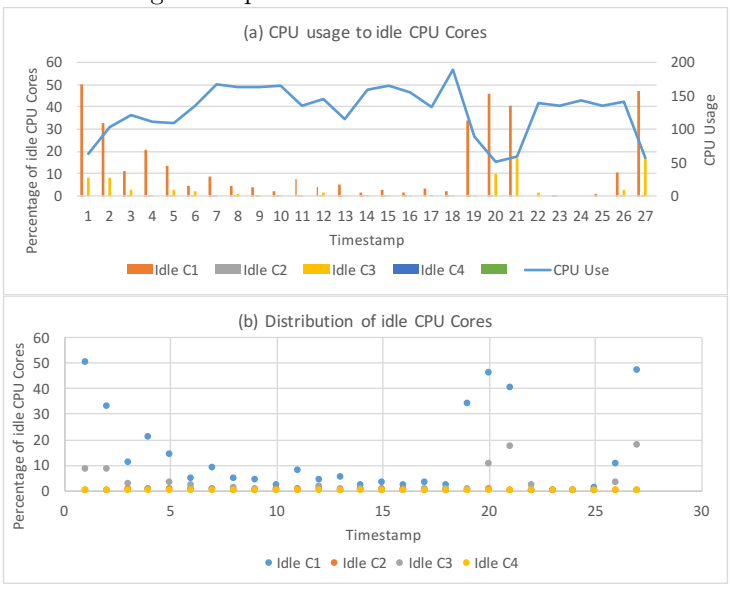
We further explore the CPU resource consumption that is related with the energy consumption rates. Figure 4 we demonstrate different measurements when we run the YCSB workload on a VM deployed in OpenStack. The values reported are related with the PM where the VM is deployed. In particular we can measure the following.

(a) Figure 4(a) shows the CPU usage in relation to the idle VMs. We can observe that during the workload execution the CPU usage varies, for example from timestamp 5 to 19 we execute the "insert records" phase and the CPU usage is 175% while between 22 to 26 we execute the "update records" phase the CPU usage is around 150% . Each timestamp represents a 20 a seconds sampling rate. This means that if we

run the workload seqentially, always during the "update records" phase we expect lower CPU utilization, yet overall the CPU usage is higher since the insert phase duration is higher. This validates this study assumption, that CPU utilization and consequently energy consumption rates vary during time, and can be characterized based on well known workloads.

(b) Figure 4(b) shows the distribution of idle CPU Cores (in our case the PM has 4 Cores) during the experimental study. We can observe that most of the workload execution time the idle CPU values are 0% meaning that all the cores are busy and this represents the 88% of the total samples. However, for fewer time instances, around 12% VM idle value is higher than 10%. Similar to (a), our assumption that momently scheduling decision making based on resource utilization does not represent always the overall usage. For example, if the scheduler is based on the most available CPU usage and the request comes on timetamp 20, the server could be selected for VM placement as there is an idle core (with 50% availability). However, after 20 seconds, based on the workload execution this might change.

Figure 4: (a) The CPU usage comparison to idle CPU Cores and (b) the distribution of the idle CPU Cores during the experiment



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 affect 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 place VMs in PMs that minimize the CPU steal time. In other words, VM scheduling to a host that the already placed neighbours VMs will steal the less amount of CPU from the VM to be placed.

3. 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.

3.1. 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 Lucas-Simarro et al. (2013) 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 Do et al. (2011), 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 Xi et al. (2015), 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 Bin et al. (2011) authors presented a hight 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 need to relocate other VMs. The experimental solution is based on a simulation, and the results show optimization in load balancing compared to a stand alone host solution.

In Marzolla et al. (2011) 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 exhange 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.

3.2. 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 Fang et al. (2013) 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 Pintea et al. (2012), to select the path with the sufficient capacity.

In Le et al. (2011) 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 Beloglazov and Buyya (2012), 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 Tseng et al. (2015) 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 Cardosa et al. (2009) 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 Van et al. (2010), 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 Goudarzi et al. (2012) 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 Goudarzi and Pedram (2012) 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 Dupont et al. (2012) 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 Li et al. (2009), 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 seperate their workloads to web/database server, compute-intensive and common applications.

3.3. Resource provisioning

This section summarizes the resource provisioning VM placement approaches.

In Chaisiri et al. (2012), 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 Corradi et al. (2014), 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 Van et al. (2009), 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 Tordsson et al. (2012) 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.

3.4. 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 us-

age for more efficient utilization of the PMs within a cloud cluster.

In Gao et al. (2013), 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 Kousiouris et al. (2011) 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 Mills et al. (2011), 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 Elmroth and Larsson (2009), 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 Meng et al. (2010), 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 Piao and Yan (2010), 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 Xu and

Fortes (2010), 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 Jayasinghe et al. (2011), 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 Jiang et al. (2012) 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 Biran et al. (2012), 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 Lloyd et al. (2014) 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.

3.5. OpenStack scheduling

OpenStack is a open source platform for private and public clouds (OpenStack (2016)). 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 Khedher (July 2015). 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 Khedher (July 2015) 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 AggregateCoreFilter based on CPU allocation ratios, AggregateDiskFilter based on disk allocation ratios, AggregateImagePropertiesIsolation for determine images that are matched with aggregation metadata and many others as presented in (OpenStack (2016)).

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 Intel-opensource.org (2015) 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-commiting 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.

3.6. Review conclusions

Figure 5 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

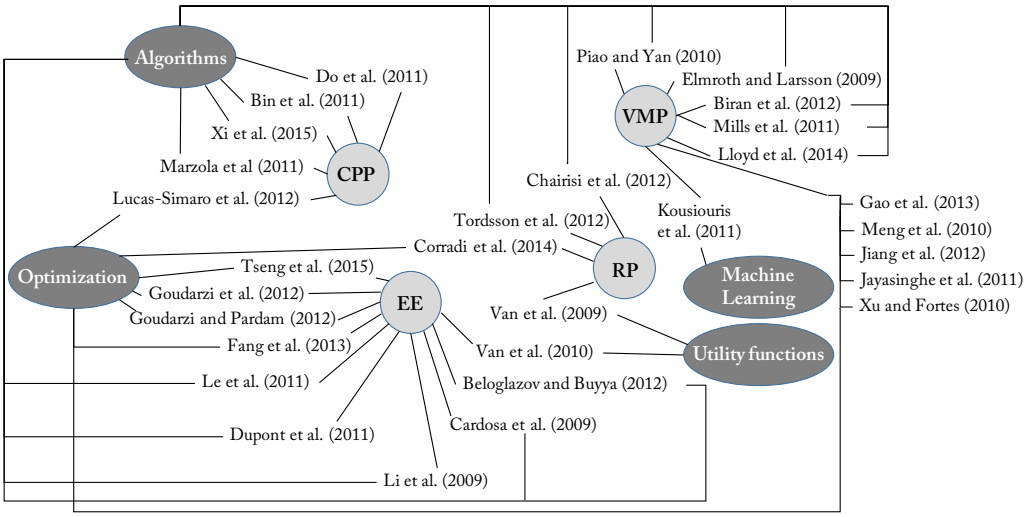


Figure 5: Classification of literature review approached according to CPP, EE, VMP and RP and according to their method (algorithms, optimization, utility functions and machine learning).

We also summarize literature review analysis in Appendicie's tables 1, 2. Most of the works like Corradi et al. (2014), Cardosa et al. (2009), , Van et al. (2010), Bin et al. (2011), Chaisiri et al. (2012), Beloglazov and Buyya (2012), , Biran et al. (2012), Intel-opensource.org (2015) and Goudarzi and Pedram (2012) focus on algorithms and optimization methodologies to achieve better VM placement in cloud systems using either current resource usage or historical data from cluster PMs. Works like Jayasinghe et al. (2011), Dupont

et al. (2012), Lucas-Simarro et al. (2013) and Van et al. (2009) focus on constraints satisfactin objectives following PMs resource usage. Approaches like Gao et al. (2013), Tseng et al. (2015) and Li et al. (2009) focus on optimizing NP hard VM placement. Approaches like Fang et al. (2013), Jiang et al. (2012), Jayasinghe et al. (2011) and Meng et al. (2010) 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 Kousiouris et al. (2011) 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. We anticipate that learning from past VM experiences could assist on a more refined VM scheduling and placement in cloud systems. This could highlight significant benefits including the following.

1. Sophisticated pattern recognition of already executed VMs will allow classification of their type and quantification of utilization levels.

2. Intelligent decision making based on the most important features predicted on real time.

3. Self modifying and auto-tweaking according to the on the fly training set by extracting knowledge from data and train according to everyday behaviour.

Cloud platforms use simple weighting schedulers based only on PM resource utilization. Authors in Intel-opensource.org (2015) 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 Folco (2015). This overloads powerful PMs in the stack and leaves low RAM PMs under-utilized.

In this work, we propose an algorithm for VM placement according to other VMs resource utilization. We identify the literature gap with regards to the *monitoring of real time data and using past VM usage to make future decisions at the time that data is produced using machine learning models*. In addition over-committing of VMs could be further optimized by exploring the real VM resource usage pattern, rather then the prospective one.

4. VM scheduling based on virtual resource usage

This section presents the VM scheduling algorithm that includes a number of optimization schemes based on machine learning. The algorithm optimizes the VM selection phase by allowing real time monitoring and analytics of physical and virtual resources.

4.1. VM scheduling in Clouds

This work is based on the OpenStack VM scheduling process. The assumption is that requests for VMs are submitted to the Cloud, thus a process is initiated in order to search for the best PMs in the cluster that satisfies the selected criteria. Finally the VM is scheduled for placement to the best ranked PM. In our case, we extend the selection step to include online scheduling. Figure 6 demonstrates the VM scheduling process that includes the following.

1. The cloud hosting service receives a request for creating a new VM including resources to be allocated such as number of Cores, memory and hard disk resources.

2. The service sends a request to the database service to record the operation.

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

4. The scheduler performs the following two subprocesses.

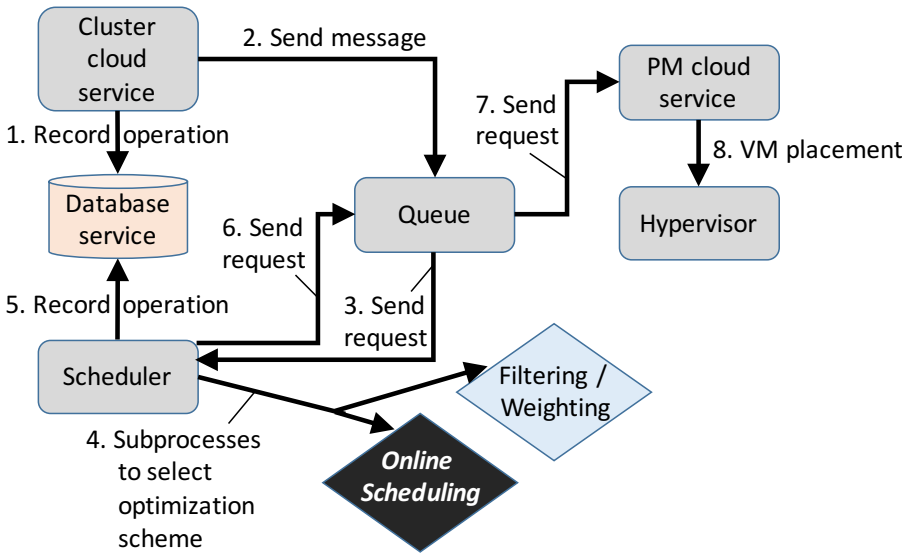


Figure 6: Scheduling VMs in Cloud systems

(a) First it collects a list with all PMs of the cluster and performs a filtering by selecting PMs according to available resources. The filter is binary, meaning that a PM is capable or not for hosting a VM.

(b) Secondly, it performs a weighting of selected PMs according to different features (i.e. available RAM) and sorts a new list with ranked hosts. This allows prioritization of one weigher against the other and then it selects the first one in the list. Formula 1) (OpenStack (2016)) 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.

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

The online scheduling process comes to enhance the filtering and weighting scheduling by including more refined criteria such as past resource usage over time.

5. The scheduler sends a request to the database service to record the operation.

6. The scheduler sends a request to the queue of the cloud hosting service with the selected PM for placement.

7. The service sends a request to the cloud hosting service of the selected PM.

8. The selected PM hypervisor launches the VM and the operation is recorded to the database service.

The online scheduler includes selection criteria that incorporate information from (a) *PM usage PMU* that is based on instant data collection, (b) *PM analytics (PMA)* and (c) *VM analytics (VMA)*. In (b) and (c) we use machine learning to define utilization levels over a period of time. Next we define the selection criteria.

1. *PMU*: As discussed before, a typical cloud environment includes filtering of PMs according to required computational resources (f) and weighting (w) according to a selected feature (i.e. current available RAM at the time of placement). In this case PMs are ranked and sorted so the scheduler selects the one with the most available resources (e.g. RAM). Most of the literature approaches focused on this level of scheduling that regards to more efficient PM evaluation.

2. *PMA*: This process follows the traditional approach in order to filter and weight PMs, however 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. In detail, the algorithm evaluates past PMs 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 PMs are ranked accordingly. In detail, by using this algorithm PMs are re-ranked according to a machine learning classifier. For example we use as *data set* resource information from 24 hours monitoring and as *training set* a seven day resource usage monitoring.

3. *VMA*: This process is similar to previous case, however it includes data analytics for already deployed VMs. In particular, each PM has a list

of VMs instantiated within, thus the algorithm classifies VM usage and provides an aggregated view of the whole VM cluster per PM. At the end PMs are ranked according to the average value of their VMs resource usage for selected parameters (e.g. CPU, Memory etc.). The end of this process is PM ranking according to the VM usage per timeframe window.

Finally the algorithm evaluates the selected criteria as in *a,b,c* and concludes to the best available PM for VM placement. The monitoring engine allows online resource usage monitoring data collection from PMs or VMs. The engine is capable of collecting system data on interval and stores data to an online cloud service that makes it available for data processing. Data is collected every a tiny time interval (e.g. 1 second) 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.

4.2. Algorithm

We study the problem of VM scheduling in PMs in a Cloud system. We use k-nearest neighbour learning and support vector machines (SVM) methods. The *k-nearest neighbour learning* method to predict the value of new points according to their distance to a predefined number of training samples. The algorithm includes the following. First it determines parameter k by training the dataset. The algorithm gives the option to train according to the same data set, to train and split and to perform k-fold cross validation. Then it calculates the Euclidean distance between the query data (i.e. the resource utilization status) and the training set (i.e. CPU user, nice, system, idle and percent). Then it sorts the distance and determine the nearest neighbour according to k and gathers the nearest neighbours of the category. Finally, it predicts the query instance (in our case the resource utilization status) according to the majority of the category. KNN predictions is the av-

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

erage of the k-nearest neighbours outcome given by the function $y = \frac{1}{K}\sum_{i=1}^{k} y_i$. The *support vector machines (SVM)* method for binary classification according to resource utilization status. The algorithm includes the following. Lets consider the example dataset of CPU(CPU user, nice, system, idle and 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 margin and we train SVM by minimizing the error function given by the formula $\frac{1}{2}w^T w + C\sum_{1}^{n} a_i$, where C is the capacity constraint, a the parameters for handling no seperable data and n are the training data.

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 $R\nu = \begin{bmatrix} c\nu & m\nu & d\nu \end{bmatrix}$ 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 $M = \begin{array}{c} \\ p_1 \\ ... \\ p_q \end{array} \begin{array}{c} \begin{array}{ccc} c & m & d \end{array} \\ \begin{pmatrix} c_1 & m_1 & d_1 \\ ... & ... & ... \\ c_q & m_q & d_q \end{pmatrix} \end{array}$ 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 a machine learning method to model the selection of PMs. We use the following methods to classify resources, according to the selected features. 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 (c) regression where the output variable takes continuous values e.g. for CPU, memory percentages etc.

First we use a *KNN* classifier that defines a training set: $(x_1, y_1), (x_2, y_2), ..., (x_w, y_w)$, where x is the selected features including 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) and y is the target feature (for example the resource usage). We assume that a d-dimensional feature vector of real numbers $x_w = (x_i^1, x_i^2, ..., x_i^d)$ for all i. Also the class label y_i is for all i. The task of this is to determine the new y_α for a given x_α. The algorithm uses the Euclidean distance $([(x_j^1 - x_\alpha^1)^2 + ... + (x_j^d - x_\alpha^d)]$ to calculate the k closest distance and classifies by y^{knn}. To measure the test set accuracy we use the $k_acc = (\sum y_i^k nn)/m$ where m is the new data $(x_{n+1}, ..., x_{n+m}$ and $y_{n+1}^k nn, ..., y_{n+m}^k nn)$. Further, we perform k-fold cross validation for the data set for more accurate results.

Secondly we use an SVM model, where we have a given dataset $D = (x_1, y_1), (x_2, y_2), ..., (x_w, y_w)$ where $x_i = (x_{i1}, x_{i2}, ..., x_{i\nu})$ 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 SVM linear function as follows $f(x) = \sum_{i=1}^{\nu}(w_i \times x_i) + \beta$ where w is the weight vector $w = (w_1, w_2, ..., w_r)$ and β is bian If $f(x_i) \geq 0$ then the vector $x_i \geq 0$ and y_i

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

SVM chooses a hyperplane that maximizes the margin between points, thus the distance between two their distance will be the shortest to the closest positive and negative data distance. We define h_+ and h_- two parallel hyperplanes and by altering the w we obtain

$$h_+ : \sum_{i=1}^{\nu}(w \times x_+) + \beta = 1 \mid h_+ : \sum_{i=1}^{\nu}(w \times x_-) + \beta = -1$$

and $\sum_{i=1}^{\nu}(w_i \times x_i) + \beta \geq 1$ and $\sum_{i=1}^{\nu}(w_i \times x_i) + \beta \leq 1$. Thus for an x_s belonging to the hyperplane (x_s, y_s) its distance to $h_+$is calculated as $d_+ = \frac{|\sum_{i=1}^{\nu}(w \times x_+) + \beta - 1|}{\|w\|} = \frac{1}{\|w\|}$.

The algorithm calculates the model accuracies according to the incoming data per interval cl_int that is measured in seconds. Thus, every cl_int, we measure the machine learning accuracy indicators (in our case KNN_acc, SVM_acc) and we select the best method with the highest accuracy for classification and regression models respectively. Then the algorithm evaluates the result, selects the best one with the highest accuracy and sets this as default for the prediction phase. The output of the algorithm predicts resource usage (e.g. utilziation levels) for the selected features (i.e. cpu percent and memory percent). We set the algorithm to the following configurations.

1. Filter & weighing according to PM resource utilization levels given by $u = \frac{\gamma}{\tau}$ where γ is the actual resource usage and τ the total.
2. PMA and VMA based on the *KNN* and *SVM* models of the machine learning engine. The algorithm selects the model with the highest accuracy.
3. The algorithm creates a new vector $V = \begin{matrix} \\ WF \\ PPML \\ VMML \end{matrix} \begin{matrix} PM \\ \begin{pmatrix} u_{p_\kappa} \\ u_{p_\lambda} \\ u_{p_\mu} \end{pmatrix} \end{matrix}$ where $p_\kappa, p_\lambda, p_\mu \in P$.
4. The algorithm sets three weights w_WF, w_PMA, w_VMA where $w_WF + w_PMA + w_VMA = 1$. We set as default values 0.2, 0.6 and 0.2 respectively, thus making PMA scheduling based 60% on PMA. The algorithm returns the best PM for VM placement.

Pseudocode algorithm 1 demonstrates the process of VM scheduling solution.

5. Evaluation

The evaluation includes the following:

Algorithm 1 Scheduling algorithm

Data: P: A physical node
V: A virtual node
nc_P, cpu_V: The number of cores of P and V respectively
cpu_P, cpu_V: The CPU of P and V respectively
mem_P, mem_V: The memory of P and V respectively
hd_P, hd_V: The hd of P and V respectively
R: A request, m: A message, q: A queue
RV: A request for VM
P_{LIST}: List of physical nodes (P)
V_{LIST}: List of virtual nodes (V)
P_{SEL}: The selected P from P_{LIST} to host the RV
db: The database for storing operations
s: The scheduler component, eval: A Boolean variable
f: A filter, w: A weight, norm: A normalization cloud function
RunSel: A P selector function
PPMLEngine, VPMLEngine: The machine learning engine for PM and VM

Input: RV = {nc, cpu, mem, hd}
$P_{LIST} = [P_{1_{nc,\ cpu,\ mem,\ hd}}, P_{n_{nc,\ cpu,\ mem,\ hd}}]$
$V_{LIST} = [V_{1_{nc,\ cpu,\ mem,\ hd}}, V_{n_{nc,\ cpu,\ mem,\ hd}}]$

Output: P_{SEL}

procedure MACHINE LEARNING VM PLACEMENT(*Req*)
  db ←O ▷ Record operation to the database
  m[RV] ▷ Forms the message according to the request
  q[*i*] ←m[RV] ▷ Sends message to the queue
  if $q_{=}i$ **then**
    s ←m[RV] ▷ Sends message to the scheduler
    for `each P in` P_{LIST} **do**
      `get(nc, cpu, mem, hd)`
      P_{SEL}[] ←eval
      for `each P in` P_{SEL}`[]` **do**
        w ←$w_{P_{SEL}}$ + norm($w_{P_{SEL}}$)
      `PMA()`
      for `each V in` V_{LIST} **do**
        VMA([P_{SEL}])
    db ←O ▷ Record operation to the database

16: **procedure** (*continues*)
17: P_{SEL} ←RunSel(P_{SEL}[],PPML[P_{SEL}],VPML[P_{SEL}]) ▷ Select P
18: **return** P_{SEL} ▷ The selected host
19: $q[i]$ ←m[P_{SEL}] ▷ Sends message to the queue
20: $PSel$ ←RV ▷ Sends message to the compute node
21: $P_{\text{SEL}_{\text{hyper}}}$ ←RV ▷ Launch RV to hypervisor

1. Experimental analysis of the real time classifier.

2. Experimental analysis linear regression model for Apache Cassandra, MongoDB and Elasticsearch systems.

3. Experimental analysis of OpenStack actions during workload execution.

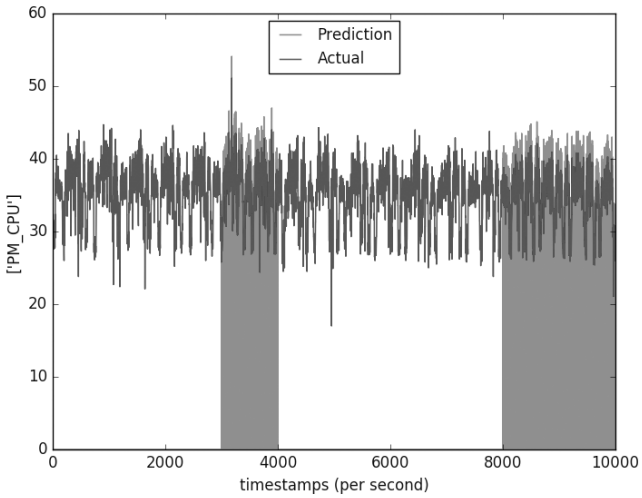


Figure 7: Scheduling VMs in Cloud systems

6. References

Beloglazov, A., Buyya, R., Sep. 2012. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr.

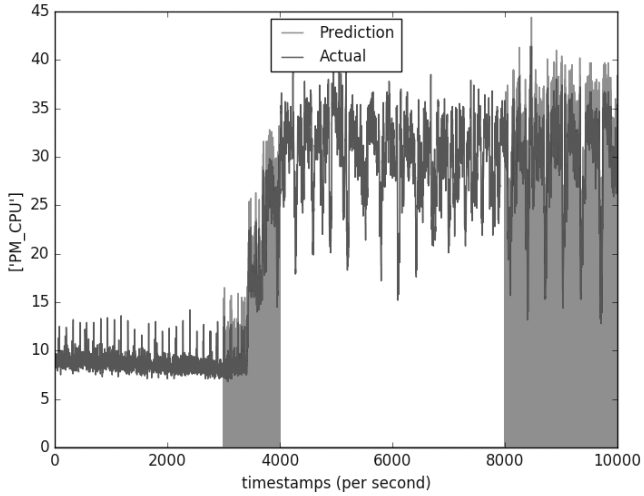


Figure 8: Scheduling VMs in Cloud systems

Comput. : Pract. Exper. 24 (13), 1397–1420.
URL http://dx.doi.org/10.1002/cpe.1867

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

Biran, O., Corradi, A., Fanelli, M., Foschini, L., Nus, A., Raz, D., Silvera, E., 2012. 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). CCGRID '12. IEEE Computer Society, Washington, DC, USA, pp. 498–506.
URL http://dx.doi.org/10.1109/CCGrid.2012.119

Cardosa, M., Korupolu, M. R., Singh, A., 2009. 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. IM'09. IEEE Press, Piscataway, NJ, USA, pp. 327–334.
URL http://dl.acm.org/citation.cfm?id=1688933.1688986

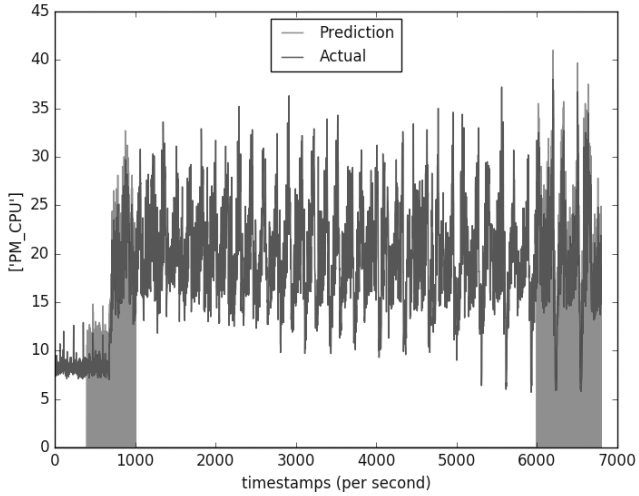


Figure 9: Scheduling VMs in Cloud systems

Chaisiri, S., Lee, B. S., Niyato, D., April 2012. Optimization of resource provisioning cost in cloud computing. IEEE Transactions on Services Computing 5 (2), 164–177.

Corradi, A., Fanelli, M., Foschini, L., Mar. 2014. Vm consolidation: A real case based on openstack cloud. Future Gener. Comput. Syst. 32, 118–127. URL http://dx.doi.org/10.1016/j.future.2012.05.012

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

Dupont, C., Schulze, T., Giuliani, G., Somov, A., Hermenier, F., 2012. 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. e-Energy '12. ACM, New York, NY, USA, pp. 4:1–4:10.
URL http://doi.acm.org/10.1145/2208828.2208832

Elmroth, E., Larsson, L., Aug 2009. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds. In: Grid and Cooper-

ative Computing, 2009. GCC '09. Eighth International Conference on. pp. 253–260.

Fang, W., Liang, X., Li, S., Chiaraviglio, L., Xiong, N., 2013. Vmplanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers. Computer Networks 57 (1), 179 – 196.

Fitfield, T., Nov. 2013. Introduction to openstack. Linux J. 2013 (235). URL http://dl.acm.org/citation.cfm?id=2555789.2555793

Folco, R., 2015. Openstack nova scheduler: Disable ram weigher.

Gao, Y., Guan, H., Qi, Z., Hou, Y., Liu, L., Dec. 2013. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J. Comput. Syst. Sci. 79 (8), 1230–1242. URL http://dx.doi.org/10.1016/j.jcss.2013.02.004

Goudarzi, H., Ghasemazar, M., Pedram, M., May 2012. 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. pp. 172–179.

Goudarzi, H., Pedram, M., 2012. Energy-efficient virtual machine replication and placement in a cloud computing system. In: Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing. CLOUD '12. IEEE Computer Society, Washington, DC, USA, pp. 750–757. URL http://dx.doi.org/10.1109/CLOUD.2012.107

Intel-opensource.org, 2015. Utilization based scheduling in openstack compute (nova).

Jayasinghe, D., Pu, C., Eilam, T., Steinder, M., Whally, I., Snible, E., July 2011. 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. pp. 72–79.

Jiang, J., Lan, T., Ha, S., Chen, M., Chiang, M., March 2012. Joint vm placement and routing for data center traffic engineering. In: INFOCOM, 2012 Proceedings IEEE. pp. 2876–2880.

Khedher, O., July 2015. Mastering OpenStack. Vol. 978-1-78439-564-3. Packt Publishing.

Kousiouris, G., Cucinotta, T., Varvarigou, T., Aug. 2011. 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), 1270–1291.
URL http://dx.doi.org/10.1016/j.jss.2011.04.013

Le, K., Bianchini, R., Zhang, J., Jaluria, Y., Meng, J., Nguyen, T. D., 2011. 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. SC '11. ACM, New York, NY, USA, pp. 22:1–22:12.
URL http://doi.acm.org/10.1145/2063384.2063413

Li, B., Li, J., Huai, J., Wo, T., Li, Q., Zhong, L., Sept 2009. Enacloud: An energy-saving application live placement approach for cloud computing environments. In: Cloud Computing, 2009. CLOUD '09. IEEE International Conference on. pp. 17–24.

Lloyd, W., Pallickara, S., David, O., Arabi, M., Rojas, K., March 2014. Dynamic scaling for service oriented applications: Implications of virtual machine placement on iaas clouds. In: Cloud Engineering (IC2E), 2014 IEEE International Conference on. pp. 271–276.

Lucas-Simarro, J. L., Moreno-Vozmediano, R., Montero, R. S., Llorente, I. M., Aug. 2013. Scheduling strategies for optimal service deployment across multiple clouds. Future Gener. Comput. Syst. 29 (6), 1431–1441.
URL http://dx.doi.org/10.1016/j.future.2012.01.007

Marzolla, M., Babaoglu, O., Panzieri, F., June 2011. Server consolidation in clouds through gossiping. In: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a. pp. 1–6.

Meng, X., Pappas, V., Zhang, L., 2010. Improving the scalability of data center networks with traffic-aware virtual machine placement. In: Proceedings of the 29th Conference on Information Communications. INFOCOM'10. IEEE Press, Piscataway, NJ, USA, pp. 1154–1162.
URL http://dl.acm.org/citation.cfm?id=1833515.1833690

Mills, K., Filliben, J., Dabrowski, C., 2011. Comparing vm-placement algorithms for on-demand clouds. In: Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science. CLOUDCOM '11. IEEE Computer Society, Washington, DC, USA, pp. 91–98.
URL http://dx.doi.org/10.1109/CloudCom.2011.22

OpenStack, 2016. Scheduling.

Piao, J. T., Yan, J., 2010. 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. GCC '10. IEEE Computer Society, Washington, DC, USA, pp. 87–92.
URL http://dx.doi.org/10.1109/GCC.2010.29

Pintea, C.-M., Pascan, C., Hajdu-Macelaru, M., 2012. Comparing several heuristics for a packing problem. CoRR abs/1210.4502.

Tordsson, J., Montero, R. S., Moreno-Vozmediano, R., Llorente, I. M., Feb. 2012. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Gener. Comput. Syst. 28 (2), 358–367.
URL http://dx.doi.org/10.1016/j.future.2011.07.003

Tseng, F.-H., Chen, C.-Y., Chou, L.-D., Chao, H.-C., Niu, J.-W., Oct. 2015. Service-oriented virtual machine placement optimization for green data center. Mob. Netw. Appl. 20 (5), 556–566.
URL http://dx.doi.org/10.1007/s11036-015-0600-9

Van, H. N., Tran, F. D., Menaud, J. M., Oct 2009. Sla-aware virtual resource management for cloud infrastructures. In: Computer and Information Technology, 2009. CIT '09. Ninth IEEE International Conference on. Vol. 1. pp. 357–362.

Van, H. N., Tran, F. D., Menaud, J. M., July 2010. Performance and power management for cloud infrastructures. In: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on. pp. 329–336.

Xi, S., Li, C., Lu, C., Gill, C. D., Xu, M., Phan, L. T. X., Lee, I., Sokolsky, O., June 2015. Rt-open stack: Cpu resource management for real-time cloud

computing. In: Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on. pp. 179–186.

Xu, J., Fortes, J. A. B., 2010. Multi-objective virtual machine placement in virtualized data center environments. In: Proceedings of the 2010 IEEE/ACM Int'L Conference on Green Computing and Communications & Int'L Conference on Cyber, Physical and Social Computing. GREENCOM-CPSCOM '10. IEEE Computer Society, Washington, DC, USA, pp. 179–188.
URL http://dx.doi.org/10.1109/GreenCom-CPSCom.2010.137

Appendix A

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

Class	**Authors**	**Method**	**Method concept**	**Parameter 1**
CPP	Do et al. (2011),	Algorithms	Canonical correlation analysis	Scheduling
CPP	Bin et al. (2011)	Algorithms	k-resilient	Load balancing
CPP	Xi et al. (2015)	Algorithms	Mapping strategy	Resource utilization
CPP	Marzolla et al. (2011)	Algorithms	Message exchanging	Resource allocation
CPP	Lucas-Simarro et al. (2013)	Optimization	Constraints satisfaction	Resource utilization
EE	Dupont et al. (2012)	Algorihtms	Constraints satisfaction	Energy consumption, CO_2
EE	Le et al. (2011)	Algorihtms	Heuristics	Electricity price, Temperature
EE	Beloglazov and Buyya (2012)	Algorihtms	Heuristics	Resource usage
EE	Li et al. (2009)	Algorithms	Heuristics (FirstFit/BestFit)	Energy consumption
EE	Cardosa et al. (2009)	Algorithms	Min-max	Power performance
EE	Fang et al. (2013)	Optimization	Greedy Bin-Packing	Traffic volume
EE	Goudarzi and Pedram (2012)	Optimization	Local search	Energy consumption
EE	Goudarzi et al. (2012)	Optimization	Probabilistic Analysis	Resource usage
EE	Tseng et al. (2015)	Optimization	Tree & Forest	Commun-ication cost
EE	Van et al. (2010)	Utility function	Constraints satisfaction	Energy consumption, profit

Appendix B

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

Class	Authors	Method	Method concept	Parameter 1
RP	Tordsson et al. (2012)	Algorihtms	Cloud brokering	Load balancing
RP	Chaisiri et al. (2012)	Algorihtms	Stochastic programming model	On-demand costs
RP	Corradi et al. (2014)	Optimization	Optimize resource usage	Resource usage
RP	Van et al. (2009)	Utility function	Constraints satisfaction	Resource usage
VMP	Piao and Yan (2010)	Algorithms	Analytical model	Network factors
VMP	Elmroth and Larsson (2009)	Algorithms	Cross VM management	Resource utilization
VMP	Biran et al. (2012)	Algorithms	Min Cut Ratio-aware	Communication cost, Resilience
VMP	Mills et al. (2011)	Algorithms	Objective method	Resource utilization
VMP	Lloyd et al. (2014)	Algorithms	Services	VM type Scalability
VMP	Kousiouris et al. (2011)	Machine learning /Algorithms	Linear regression/ Genetic algorithms	Prediction degradation
VMP	Gao et al. (2013)	Optimization	Ant colony optimization	Resource wastage, performance
VMP	Meng et al. (2010)	Optimization	Approximation	Communication cost
VMP	Jiang et al. (2012)	Optimization	Approximation (Markov)	Traffic cost
VMP	Jayasinghe et al. (2011)	Optimization	Constraints satisfaction	Server rack assignment
VMP	Xu and Fortes (2010)	Optimization	Multi-objective	Scalability, Robustness