A Sensor-Actuator Model for Data Center Optimization

Jakub Krzywda, Per-Olov Östberg, and Erik Elmroth
Department of Computing Science, Umeå University
SE-901 87 Umeå, Sweden
Email: {jakub, p-o, elmroth}@cs.umu.se

***Abstract*—Cloud data centers commonly use virtualization technologies to provision compute capacity with a level of indirection between virtual machines and physical compute resources. In this paper we explore the use of that level of indirection as a means for autonomic data center configuration optimization and propose a sensor-actuator model to capture optimization-relevant relationships between data center events, monitored metrics (sensors data), and management actions (actuators). The model quantifies and characterizes a wide spectrum of actions to help identify the suitability of different actions in specific situations, and outlines what (and how often) data needs to be monitored to capture, classify, and respond to events that affect the performance of data center operations. To support the analysis and illustrate the utility of the model, the paper also details a set of testbed experiments aimed to characterize trade-offs in the use of different actions in infrastructure optimization.**

I. Introduction

Infrastructure-as-a-Service (IaaS) cloud computing environments aggregate compute resources in data centers and provision compute capacity to end-users via service-based interfaces. To facilitate a flexible and resource-agnostic view of compute resources, these environments typically abstract physical resources using some form of virtualization technology, e.g., virtual machines or process containers, and provide metered services for some form of virtual resources [1].

In addition to providing a flexible and resource-agnostic model for provisioning of resources [2], this use of virtualization technology also provides a level of indirection between physical and virtual resources that can be used to dynamically optimize the deployment and configuration of data center resources [3]–[5]. In this work we take the perspective that this optimization should be performed in an autonomic manner, with an optimization controller that continuously monitors and analyses the state of the data center, and plans and suggests optimization actions that can be executed automatically by software components (controllers) or in a semi-automated manner by system administrators [6]. In autonomic systems, this is commonly formulated as a Monitor-Analyze-Plan-Execute-Knowledge (MAPE-K) loop [7], where the optimizer encompasses the analysis and planning steps.

In order to realize such an optimizer, a model with a holistic view of the data center, which includes knowledge of not only the state of the (virtual and physical) resources but also of data center events, contextual information of resources, availability schedules, etc. is needed. In this work we address construction of a sensor-actuator model (encompassing the monitor and execute parts of the MAPE-K loop) for autonomic data center optimization, and characterize the data (sensors) and actions (actuators) that can be used in dynamic optimization of data center configurations. We investigate and classify various planned, predicted, and unpredicted events that can occur in data center operation, both external (e.g., changes in the incoming workload) and internal (e.g., hardware failures or changes in resource availability schedules), and map these to software-defined management actions that can be taken to respond to events and steer the data center towards a (at policy level defined) desired state.

As management actions have widely varied characteristics in dimensions such as performance (e.g., the time needed to apply actions or the performance overhead induced by actions), impact (e.g., how much a certain action can be expected to alter the performance of a resource or application), and applicability (e.g., whether an action is suitable in a specific situation), we attempt to build a taxonomy for sensors and actuators in data center configuration optimization, and outline the relationships between these to construct a knowledge base for automated data center optimization.

The main contributions of this paper are:

- A classification and characterization of planned, predicted, and unpredicted events that may occur in data center operations (Section IV).
- An analysis of the spectrum of actuators that are available to optimizers in current data centers, and a discussion of the applicability and suitability of different actions in handling of specific events (sections V and VI).
- An analysis of what (and how often) data needs to be monitored by sensors to facilitate detection and prediction (based on workload and resource models) of data center events (Sections VII).

The rest of the paper is organized as follows. Section II outlines the interpretation of the MAPE-K loop used in this work, and Section III samples related work. Section IV presents the proposed classification of data center events and Section V outlines an identified set of data center configuration parameters that can be adjusted to respond to events. Section VI discusses actuators that can be used to modify data center configuration parameters and Section VII describes the data requirements for mapping actuators to events. Section VIII specifies the architecture of the proposed sensor-actuator model and Section IX concludes the paper.

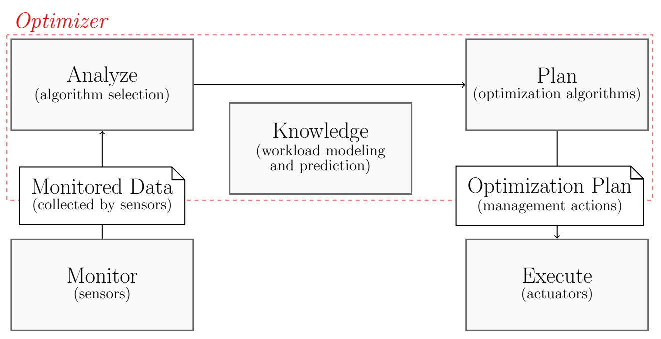

Fig. 1. The sensor-actuator model and its mapping to the MAPE-K loop.

II. MAPE-K Loop

The concept of autonomic computing defines systems that manage themselves following guidelines set by system administrators [7]. The primary building block of autonomic systems is the *autonomic element*, a component that consists of two parts: a *managed element* that requires regulation, e.g., a physical machine, and an associated *autonomic manager* that controls the managed element. Typically, autonomic managers are designed as a MAPE-K loop (illustrated in Figure 1) to structure the control process. In this paper we focus on the Monitor and Execute steps of the MAPE-K loop, and the Knowledge base that relates them. To illustrate our view of the MAPE-K loop in the context of data center optimization, and how the proposed sensor-actuator model is mapped to it (also depicted in Figure 1), we first here describe our interpretation of the individual steps of the MAPE-K loop.

Monitor. The purpose of the monitor step of the MAPE-K loop is to collect information about the state of managed elements and their environments using sensors. For optimization we identify two primary goals of monitoring: data collected by sensors should facilitate: (1) detecting events that require action, and (2) creating prediction models for workload and resource consumption.

Analyze. In the analyze step, the autonomic manager decides whether the configuration of a managed element is appropriate for the workload the element is currently facing. If the element configuration is found adequate, the manager returns to the monitoring step and may schedule the next analysis after some time (periodic approach) or sets trigger conditions for specific events (event based approach). If there is room for improvement, the manager identifies what type of changes are necessary (reactive vs. proactive), at what level (e.g., physical machine, cluster, data center) changes are to be made, and what optimization algorithm should be used taking into account the scale of necessary changes and the available time for running an optimization algorithm.

Plan. During the plan step, the autonomic manger runs the optimization algorithm that was selected in the analyze step and creates an optimization plan containing a list of recommended management actions – changes to the current configuration of managed element.

Execute. In the execute step, changes suggested in the plan step are enacted. The autonomic manager executes management actions described in the optimization plan using actuators, monitors their execution, and reports the progress.

Knowledge. Data gathered during the previous steps that can facilitate future decision making processes are stored in a knowledge base. This includes, e.g., models of workloads describing changes over the time, predicted and measured costs of actions (in respect to time, performance overhead, and consumed power), characteristics of anomalies that have been observed to lead to workload spikes, and quantifications of the performance of optimization algorithms (execution time, quality of solutions, and scales of improvement).

III. Related Work

Cloud computing hypervisors such as Kernel-based Virtual Machines (KVM) [8], offer a wide variety of management actions that allow data centers to dynamically adapt infrastructure configurations using management actions (e.g., virtual machine migration, suspending physical and virtual machines, horizontal elasticity). However, most proposed solutions for optimizing data center operations focus on single management actions and fail to fully leverage the potential of selection of management actions for specific situations.

Nevertheless, several systems that use a substantial subset of available management actions exist. For example, Mistral [3] is a holistic controller framework that optimizes tradeoffs among power consumption, application performance, and adaptation costs. The Mistral controller uses the following adaptation actions to improve data center configurations: increase/decrease virtual machine CPU capacity (vertical elasticity), add/remove virtual machines (horizontal elasticity), migrate virtual machines, and power down/up physical machines. To handle large-scale infrastructures a multi-level hierarchy of controllers is introduced, where lower-level controllers manage smaller numbers of physical machines at fine-grained time granularity, while higher-level controllers coordinate the work of lower-level controllers at more coarse-grained time granularity. Mistral uses workload prediction for estimating the stability of new configurations (i.e. how long a new configuration will be beneficial), but the controller works in a reactive way and optimization is only triggered when a current configuration is already suboptimal.

Another example of a solution that combines several management actions is Quasar [4], a cluster management system that increases resource utilization while providing consistently high application performance. The system uses vertical and horizontal scaling, as well as virtual machine migration actions, to control the configuration of a data center. Quasar employs a performance-centric approach where users can express performance constraints for workloads and the exact type of constraint used depends on the type of the workload, e.g., throughput and latency for latency-sensitive applications, or execution time for distributed frameworks such as Hadoop.

For classification, some partial hierarchies of management actions exist. For example, Vaquero et al. [9] focus on scaling actions at the levels of server, network, and platform and provide a hierarchy of available management actions related to application scalability both for Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) approaches. The approach tackles such problems as implementation of controllers (per-tier controllers vs. use of per-application controllers), defining rules and policies for when and how to scale systems, and load balancing within data centers.

Approaches to quantify the impact of management actions also exist. Svärd et al. characterize various algorithms for live migration of virtual machines [10]. The paper defines five criteria for live migration: continuous service, resource usage, robustness, predictability, and transparency. In the paper, the authors compare algorithms by measuring application downtime, total migration time, and performance degradation. Additionally, Sotomayor et al. [11] propose a model for estimation of the duration of suspending and resuming virtual machines. The authors run experiments to capture the influence of virtual machine resource allocation (amount of memory, number of CPUs, and filesystem used) on the duration of actions on a testbed consisting of physical machines running under the control of the Xen hypervisor.

Some attempts to incorporate the MAPE-K loop in data center optimization also exist. Maurer et al. [12] introduce the A-MAPE-K loop that adds an additional adaptation phase to the loop to adjust it for applications in PaaS cloud computing environments. During the adaptation phase, which takes place before deployment, service level agreements (SLAs) are established and sensors for monitoring application-level key performance indicators (KPIs) are introduced. The authors present a prototype implementation of the A-MAPE-K loop for controlling the level of SLA violations and propose a two-level approach for mapping low level metrics (e.g., system up time) to parameters included in SLA (e.g., system availability).

Finally, a number of ways of constructing modular data center optimizers also exist. Jennings and Stadler [13] propose a conceptual framework for resource management in cloud environments. The framework decomposes the whole management system into smaller functional elements and classify them as the responsibilities of service providers (e.g., workload management, application demand profiling, and application elasticity) or of infrastructure providers (e.g, scheduling, monitoring, and resource utilization estimation).

IV. Data Center Events

In this section, we classify data center events, e.g., requests for a new virtual machine admission, workload fluctuations, or hardware failures, by predictability and discuss their impact on data center performance. As events may introduce changes in data center environments that negatively impact the performance of data center resources and applications, e.g., increased power consumption or violated SLAs, they need to be understood to be able to assess what adjustments should be made to data center configurations in response. While some events are hard or even impossible to predict (e.g., hardware failures), others are more likely to happen at specific time periods (e.g., daily workload patterns). As indicated in Table I, we here classify event as planned, predicted, and unpredicted. Unpredictable events have to be handled in a reactive manner, which means that a data center has to operate in a suboptimal configuration for some time. However, for predicted events, management systems can prepare or even apply adjustments in advance. Table I presents a summary with examples, appropriate management approaches, required speed of introducing changes, and permissible costs of applying changes in a form of performance overhead.

TABLE I
Events Classification

Class	Examples of events	Approach	Speed	OH
Planned	PM power down for maintenance	Proactive	Slow	High
Predicted	Daily/weekly workload patterns	Proactive	Mid	Mid
Unpredicted	Flash crowds, hardware failures	Reactive	Fast	Low

Speed – an acceptable time needed to apply changes, OH – an acceptable performance overhead due to applying changes.

A. Planned Events

Planned events, e.g., powering down physical machines for maintenance or coordinated online analysis of data from experiments with fixed start and finish times, are by nature the easiest events to handle. As the time of occurrence for planned events is known in advance, management systems can prepare for them before they occur. In most cases, the time available for determining the best combination of management actions (and applying them) for planned events is long, which allows the use of actions with longer execution time. Additionally, management actions for planned events can also be allowed to have higher performance overhead and they can be scheduled to be executed during periods of decreased workload.

B. Predicted Events

Predicted events, e.g., workload changes following a weekly pattern, can similarly to planned events be managed proactively. Actions planned to respond to predicted events may have reasonably long execution times and performance overheads, but are typically limited by how far in advance an event can be predicted. The influence of high overhead can be limited by spreading actions across longer periods of time, e.g., by using minimal transfer rates for data transmissions with fixed deadlines to avoid network congestion [14]. Other examples of predicted events are terminations of MapReduce jobs, that can be estimated based on monitoring the progress of computation [15].

Moreover, the prediction of the time when changes in resource requirements will happen and the time needed for responding to the event are both important, as they combined allow management actions to be performed just in time when needed. Such approaches minimize the time when the amount of assigned resources differs from the amount that is actually needed, which allows to utilize resources more effectively.

C. Unpredicted Events

Unpredicted events, i.e. events that may or may not be expected but cannot be detected in advance, e.g., flash crowds, hardware failures, or arrivals of new services (requests for virtual machine admissions), have to be handled reactively. As such, management actions for unpredicted events have to be fast to apply and cannot impose high performance overhead as in most cases physical machines are already overloaded due to the increased workload or reduced availability of resources.

The key to success in reacting to unpredicted events lies in early detection of extraordinary conditions (e.g., workload spikes [16] or hardware failures [17]) that endanger fulfilling

TABLE II
DATA CENTER CONFIGURATION

Type	Resource	Description	Power
PHYSICAL MACHINE			
Power Management	PM	state: on/off/power-saving modes	✓
	CPU	Dynamic Frequency Voltage Scaling	✓
VIRTUAL MACHINE			
State	-	running/paused/suspended initializing/resuming/migrating	
Allocated Resources	CPU	the number of virtual CPUs	
	RAM	size	
	Storage	size	
	Network	bandwidth	
Instances	-	the number of VM instances	
VM-TO-PM MAPPING			
VM Placement	-	affinities and anti-affinities to other VMs or PMs	✓
vCPU-to-core Mapping	CPU	the number of physical cores assigned to the virtual CPUs	
vCPU Pinning	CPU	binding vCPU to physical core	✓

Power – direct influence on the power consumption.

the expected Quality of Service (QoS) or estimating the scale of changes in resource demands (for spikes) or resource availability (for failures). To enable detection of unpredicted application and resource behavior, models are needed to distinguish extraordinary conditions from normal periodic fluctuations.

V. DATA CENTER CONFIGURATION

The configuration of a data center greatly impacts its performance and operation costs, and can be used to optimize operations and dynamically adjusted to compensate for changes induced by events. To optimize data center configurations, various parameters can be modified, e.g., the placement of virtual machines, the amount of hardware resources allocated to virtual machines, or the CPU frequency of physical machines. Here, we analyze a subset of these parameters and group them into three main categories: configuration of physical machines (PM), configuration of virtual machines (VM), and VM-to-PM mappings. Table II summarizes the information about the data center configuration parameters considered in this work and presents the types of parameters used, resources that they refer to (if applicable), descriptions with additional information, and an indication whether the parameter has a direct influence on the power consumption of the infrastructure.

The first category of parameters describes the configuration of physical machines. Physical machines can be running and hosting virtual machines, or powered down to minimize the power consumption of the infrastructure in times of decreased workload. As the process of powering up a physical machine takes a long time, physical machines can use power saving modes (known also as sleep modes) to reduce startup times. Dynamic Frequency Voltage Scaling (DFVS) can also be used to reduce the power consumption of a physical machine at the price of decreased computational performance.

The second category of parameters in Table II describes virtual machines, which can be in various states.

Running: a virtual machine is hosted on a physical machine and processing a workload.

Paused: a virtual machine is not scheduled for any CPU resource.

Suspended: a virtual machine releases both CPU and memory resources assigned to it.

Moreover, there are also special transient states, when virtual machines consume resources but their processing capabilities are limited.

Initializing: occurs at the beginning of virtual machine's lifetime, when its operating system is being loaded.

Suspending: the virtual machine is moving memory content to storage prior to entering the suspended state.

Resuming: occurs between suspended and running states during the time of loading the memory.

Migrating: a virtual machine uses the resources of two physical machines (source and target) during a migration and may experience decreased Quality of Service (QoS).

Allocated resources specify how much resource capacity, e.g., CPU cores, memory, storage, and network bandwidth, is assigned to the virtual machine. The instance parameter specifies how many copies of a virtual machines are instantiated (horizontal scaling). For multi-tier applications the number of instances should be specified for each application tier, e.g. business logic layer, or data base layer.

The third category in Table II covers the application layer parameters that specify the mapping between virtual machines and physical machines. The placement indicates on what physical machine the virtual machine runs. In heterogeneous data centers, where different physical machines consume different amounts of energy, the placement influences the power consumption. The mapping between virtual CPUs and physical cores describes how many virtual CPUs share a single physical core. The CPU pinning parameter specifies whether virtual CPUs are bound to particular physical cores, and can be used to increase the performance of certain types of computations. As CPU pinning limits the possibility of optimizing the assignment of physical cores, e.g., to use core rotations to reduce the die temperature, it can have a negative influence on the power consumption of the physical machine.

One can argue that most, if not all, of the listed parameters have at least some influence on the power consumption. For example, the allocation of more resources to a virtual machine can cause the need for powering up another physical machine (and therefore indirectly result in increased power consumption). However, we here limit our reasoning to parameters with only direct influence on data center power consumption.

VI. TAXONOMY OF ACTUATORS

Here we describe management actions that can be taken to adjust the parameters that we have analyzed in Section V in reaction to the events discussed in Section IV. We structure all considered actions into a hierarchy of actuators, as presented in Figure 2. Elements on the first level of the hierarchy relate to the main categories of data center configuration parameters presented in Table II. On the second level, there are types of management actions that correspond to the parameters of data center configuration. The third level of the hierarchy consists of the specific management actions.

When describing management actions we emphasis two important characteristics: the time needed to apply the action,

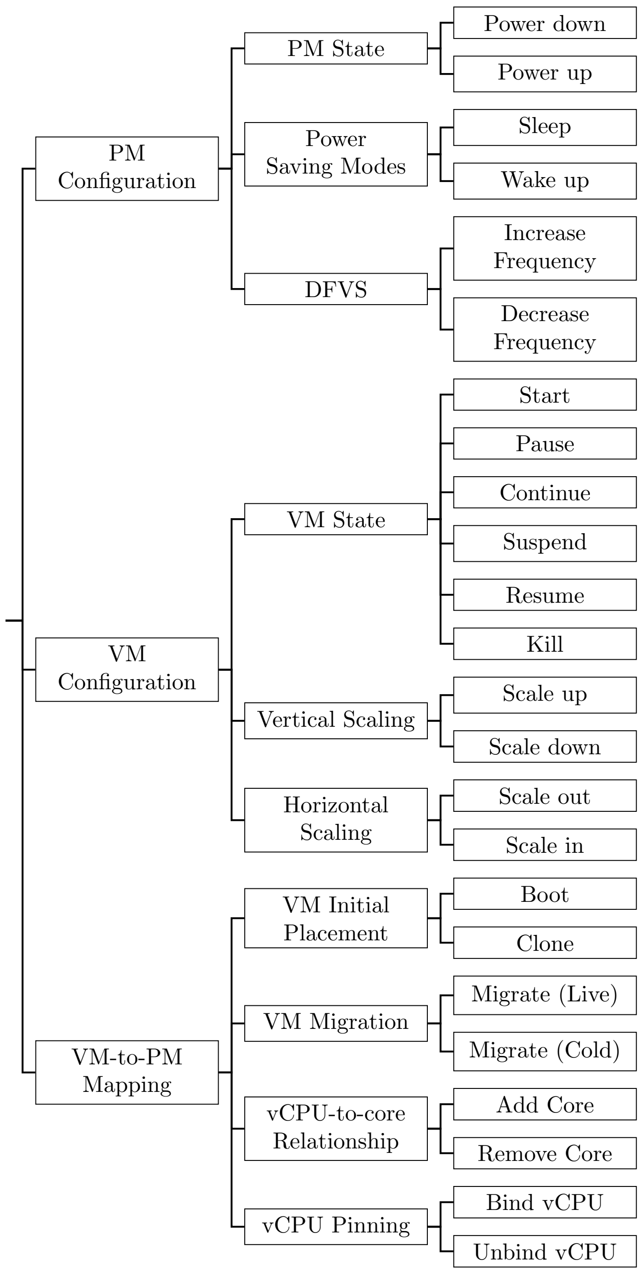

Fig. 2. The Actuators Taxonomy.

and the performance overhead that the action induces on the involved virtual and physical machines, network, and storage. These two properties are particularly important when making decision in what situation each action should be used.

We conduct several experiments to characterize various management actions. For these experiments we use a testbed consisting of HP ProLiant DL165G7 physical machines. Physical machines are equipped with 32 CPU cores (AMD Opteron(TM) Processor 6272, 2.1 GHz), 56 GB of RAM and HP SmartArray P410, 4x500 GB SATA disks arranged in RAID 1+0, and are connected with a Gigabit Ethernet network. Machines run Ubuntu 14.04 operating system and Kernel-based Virtual Machine (KVM) hypervisor. For experiments involving virtual machines we use a virtual machine running the RUBiS (Rice University Bidding System) [18].

For some of the experiments we monitor the power consumption of physical machines using AF525A, a Power Distribution Unit (PDU). The HP Intelligent Modular PDU provides per-socket power usage measurements over Simple Network Management Protocol (SNMP). To avoid capturing the overhead of monitoring in the measurements we run the script for obtaining the power consumption at a different physical machine than the one being stressed.

A. *Physical Machine Configuration*

The configuration of a physical machine can be changed by: power a physical machine up or down, putting a physical machine into a power-saving mode (sleep mode), and dynamically changing the frequency of CPU.

1) Physical Machine State: Physical machines are power down in order to reduce the total power consumption of data center when the computational resources of the physical machines are not needed [19] (i.e. when the whole workload can be handled by other physical machines in a data center). Another reason for power down physical machines is to allow maintenance work, e.g., changing a broken hard disk or upgrading the operating system. If a physical machine is not empty, i.e. hosts virtual machines, it is necessary to migrate these virtual machines to other physical machine(s) before powering down the physical machine. Moreover, power it up takes time and consumes energy during boot time even though no workload is processed. Therefore, using this management action requires reliable predictions about the future resource requirements to avoid costly oscillations. It is worth noticing, that apart from a boot time of a physical machine also instantiation or migration of virtual machines have to be performed before the workload can be processed. Ways of shortening the virtual machine instantiation time are mentioned when describing the VM Initial Placement action (Section VI-C1).

To characterize power down and power up actions we measure the time needed for gracefully shut down the operating system and power off the physical machine, and to power on the physical machine and load the operating system, respectively. We also measure the power consumption of the physical machine during these actions.

Figure 3 shows changes in the power consumption of a physical machine during the power down and power up actions. We run an idle physical machine for 20 s, then trigger power down action, next we keep the physical machine powered down for 20 s, then trigger the power up action, and finally we let the physical machine to run idle for another 20 s. The power down action takes approximately 8 s (time from sending a signal to a drop in the power consumption of physical machine). The power up action is much longer operation, it takes approximately 166 s (time from sending a signal to stabilization of the power consumption at the idle level). We observe an increase in the power consumption of the physical machine during both power down ($\sim$10%) and power up ($\sim$60%) actions comparing to the consumption of an idle physical machine. Moreover, even when powered down, the physical machine consumes some energy ($\sim$5% of the consumption of an idle physical machine). These increases in power consumption together with long time of power up action

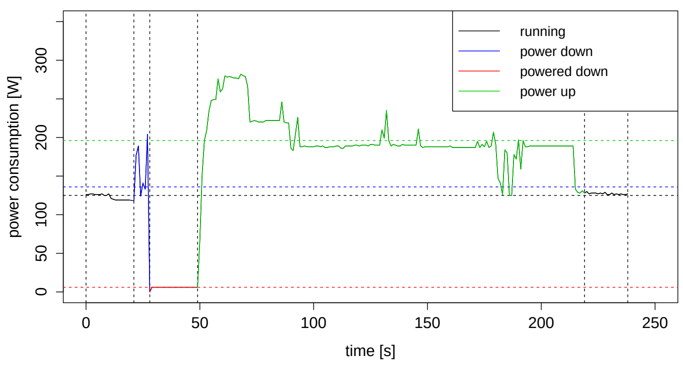

Fig. 3. Power consumption during power down and power up actions. Solid lines show measured power consumption, vertical dashed lines show the moments of transitions between states/actions, and vertical dashed lines show the average power consumption during different states/actions.

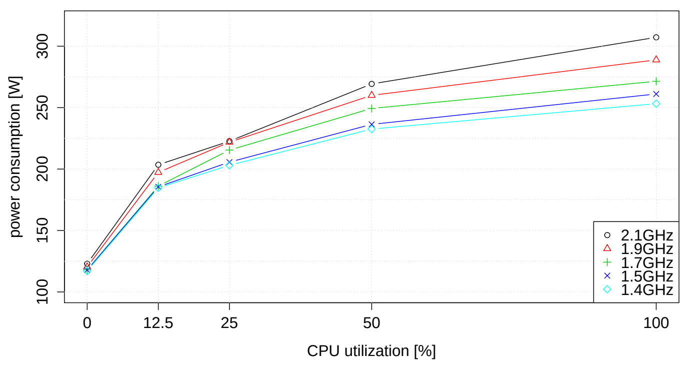

Fig. 4. Power consumption depends on CPU utilization and CPU frequency.

should be taken into account when the action of changing the physical machine state is considered. Power down physical machines for short time can paradoxically increase the power consumption of the infrastructure.

2) Power Saving Modes: Putting a physical machine into a power-saving mode is done for similar reasons as power it down – to reduce the power consumption during decreased workload [20]. The difference is that the boot time is shorter for power-saving modes [21], which can be beneficial when there is uncertainty about the demand for computing resources in the near future. The lower inertia comes with a price of a higher power consumption during the power-saving mode in comparison with powered down state.

3) Dynamic Frequency Voltage Scaling: Scaling the frequency of a physical CPU is used to change the performance capabilities and power consumption of a physical machine [22]. Scaling down the frequency, which reduces both the power consumption and performance of a physical machine, can be used during times of decreased workload to lower the operational costs of data center. Since the time necessary to scale the frequency is very short (between nanoseconds and tens of microseconds, depending on a technique used [23]), it can be used in a reactive manner to adjust to unpredicted workload changes. However, to have that possibility, it is necessary to run at least part of CPUs at a reduced frequency in a normal circumstances and use the maximum frequency to handle workload bursts.

To characterize the DFVS action we quantify the influence of both CPU utilization and CPU frequency on power consumption by measuring the power consumed by a physical machine while stressing the CPU and scaling CPU frequency. We test five CPU utilization levels: 0% (0 cores), 12.5% (4 cores fully utilize and the rest idle), 25% (8 cores), 50% (16 cores), and 100% (32 cores); at five available CPU frequencies: 1.4 GHz, 1.5 GHz, 1.7 GHz, 1.9 GHz, and 2.1 GHz. For each of 25 settings (pairs of CPU utilization level and CPU frequency) we monitor the power consumption of a physical machine for 5 seconds what gives at least 500 power consumption measurements per each setting. Values presented in Figure 4 are the average values for these measurements.

Figure 4 shows that the difference in power consumption

TABLE III
COMPARISON OF PHYSICAL MACHINE CONFIGURATION ACTIONS

Action	Time of transition	Power saving
Power up/down	166 s (up), 8 s (down)	∼95%
DVFS	10 ms (per core)	5–18%

among available CPU frequencies grows with the increase of CPU utilization. An idle physical machine running with the lowest frequency (f=1.4 GHz) consumes approximately 95% of the energy consumed by a physical machine operating at the highest frequency (f=2.1 GHz). For a fully utilized physical machine the difference is bigger, running a physical machine at the frequency of f=1.4 GHz saves around 18% of energy when compared to a physical machine operating at the frequency of f=2.1 GHz. We also observe that the relation between CPU utilization and power consumption is not linear. Running physical machines at low CPU utilization level is much less power efficient than running it at the higher CPU utilization level. That observation can be used when constructing an optimization algorithm that can target the most power efficient CPU utilization level. Moreover, our measurements show that the whole action of scaling the CPU frequency takes on average 10 ms for each CPU core, and changes to multiple cores have to be applied in a sequential manner. The time needed to apply the change is several orders of magnitude longer that what reported in literature [23], because apart from the hardware operation the whole chain of software calls has to be handled.

Table III compares times of transition and approximated power savings for powering down and up physical machines, and DFVS actions.

B. Virtual Machine Configuration

Virtual Machine Configuration actions allow to change states of virtual machines and scale them. Scaling actions allow to adjust the computational capabilities of virtual machines to changes in workloads.

1) Virtual Machine State: Changing the state of a virtual machine allows to release some computational resources that were used by that virtual machine. Thanks to that, other applications with higher priority (e.g., batch jobs with short deadlines or services experiencing degradation of QoS due to unpredicted workload increases), can benefit from additional resources. These actions are especially useful for batch jobs

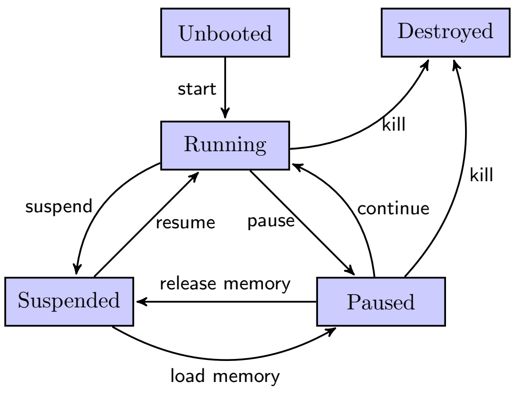

Fig. 5. Transitions between virtual machine states. Transition from suspend to destroyed state is not included because we do not consider storage as a resource that has to be released quickly.

TABLE IV
TIME OF VIRTUAL MACHINE STATE TRANSITIONS

From \ To	Unbooted	Running	Paused	Suspended	Destroyed
Unbooted	N/A	1275 ms	-	-	-
Running	-	N/A	10 ms	3444 ms	413 ms
Paused	-	160 ms	N/A	4081 ms	409 ms
Suspended	-	3199 ms	3111 ms	N/A	-
Destroyed	-	-	-	-	N/A

N/A – not applicable, "-" – transitions not considered.

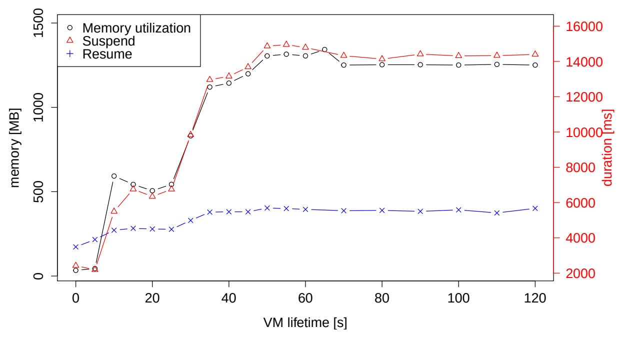

Fig. 6. Duration of suspend and resume actions (scale on the right) depends on the virtual machine memory utilization (scale on the left). When the virtual machine allocates more memory over boot up time, the duration of actions increases.

(e.g., scientific computations or MapReduce type) and services that run multiple instances of identical virtual machines (i.e. using horizontal scaling). In the case of horizontal scaling, when scaling in an application for short time (if the next workload increase is predicted to happen in the near future), instead of killing virtual machines, they can be put in a non-active state.

Various non-active states exist and differ in the resources they release and the time that is needed to make a transition between them. We here use the following terminology for non-active states: in *Paused* state a virtual machine releases only CPU resources and stores its state in a memory, while in *Suspended* state both CPU and memory are released and the state is stored to disk. Entering a non-active state enables a virtual machine to release resources and to be able to resume processing in the future from the point where it was interrupted. However, non-active virtual machines still keep some resources (disk space and possibly memory also), while not processing any workload.

Figure 5 shows virtual machine states and possible transitions between the states. At the beginning a virtual machine is *Unbooted*. When started it goes to a *Running* state. Then, a virtual machine can freely change states between *Running*, *Paused*, and *Suspended*. *Running* or *Paused* virtual machine can be killed and become *Destroyed*. Table IV compares the times of these transitions. Measurements were taken for a virtual machine running RUBiS application, with allocated 32 CPU cores and 8 GB of memory.

When a virtual machine is suspended, its memory is transferred to the storage. Therefore the duration of *Suspend* action is proportional to the amount of memory utilized by a virtual machine, as presented in Figure 6. Black circles show the changes in the memory utilization over a virtual machine lifetime. After the start of the virtual machine, memory utilization grows with a varying rate for around 65 seconds, to finally stabilze till the end of experiment. As expected, the duration of *Suspend* action, depicted as red triangles, follows the changes in the memory utilization closely. Similarly, when the virtual machine is resumed, its memory is loaded to the main memory from the disk. Thus, also the duration of *Resume* action, marked as blue crosses, depends on the amount of utilized memory. However, the changes are not as significant as for *Suspend* action.

For this experiment we use a virtual machine running the RUBiS application with 32 CPUs and 8 GB of memory allocated. We start a virtual machine, and then every 5 s we measure the memory utilization, suspend the virtual machine, and resume it. After 60 s we increase the time between measurements to 10 s, because it reaches steady state regarding the memory utilization. Experiment finishes after 120 s when we kill the virtual machine. Memory utilization is measured using `ps -u libvirt-qemu -o rss` command, which reports resident set size (the portion of memory occupied by a process that is held in main memory) of the libvirt process that contains the virtual machine. Duration of actions is measured using the `-t` option of virsh program, which outputs elapsed time information for each command.

We observe that the memory utilization of a virtual machine depends on the amount of memory allocated to it, what can be colloquially described as "have more, use more", as shown in Figure 7. Therefore, the duration of *Suspend* action depends indirectly on the amount of memory allocated to the virtual machine.

According to our measurements, the duration of *Kill* action does not depend on the memory utilization or the amount of memory allocated to the virtual machine. *Kill* action on average takes 413.36 ms with standard deviation equal to 0.92 ms. This means that whole memory allocated to a virtual machine can be released in a short time. The cost of *Kill* action is the complete loss of results computed so far, but still, that operation can be useful in case of low priority jobs or stateless services.

2) Vertical Scaling: Vertical scaling actions allow to dynamically adjust the amount of resources assigned to already hosted virtual machines. For optimization those actions can be especially important in situation when both service providers

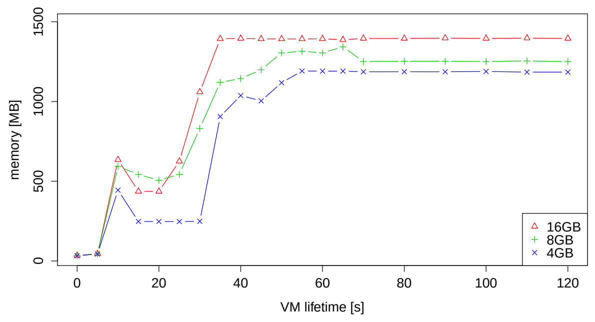

Fig. 7. Memory utilization of virtual machines depends on the amount of allocated memory.

and infrastructure providers do not know how much computational resources are needed to handle an application workload. Then, after initial allocation during initial placement, the utilization of resources and the application performance can be monitored, and appropriate changes applied to adjust the allocation to the actual needs. Moreover, scaling down actions can be used to throttle virtual machines with lower priority, e.g., batch jobs, in order to give freed resources to other virtual machines with higher priority, e.g., services suffering from unpredicted workload spikes or batch jobs with close deadline.

Vertical scaling actions can apply to different computational resources. For example, memory ballooning [24] allows to preempt memory that is not used by a virtual machine and allocate it to virtual machines that need more memory, or return it to a pool of free resources available for newly admitted virtual machines. Also the number of virtual CPUs assigned to a virtual machine can be changed in order to control the parallelism of the application (e.g., increasing the number of virtual CPUs can allow to run more threads simultaneously).

3) Horizontal Scaling: Horizontal scaling actions change the number of instances of an application or a selected tier of application [25] (e.g., logic tier in three-tier architecture). Horizontal scaling is particularly useful for applications that can benefit from distributing their workload among several workers (i.e. at least parts of them are easily paralizable).

It is important to notice that horizontal scaling actions trigger actions of other types, e.g., scaling out causes a VM Initial Placement action. In case of scaling in stateful applications, before a virtual machine is shut down, its state (e.g., sessions in progress) has to be sent to another instance. Alternatively, instead of shutting down the redundant virtual machine (*Kill* action), it can be paused or suspended and then resumed when needed again.

C. Virtual Machine-to-Physical Machine Mapping

The third category, mapping between virtual machines and physical machines, contains actions that set the allocation of virtual machines and the relationship between physical and virtual CPUs.

A service provider may want to add some restrictions regarding the allocation of his virtual machines, modeled as constraints in the optimization process [26]. VM-to-VM affinities may require two virtual machines to be placed on the same physical machine, e.g., because of substantial communication between them. Restrictions can also specify that two virtual machines should be placed on different physical machines, e.g., due to similar workload and resource needs, or because of redundancy reasons (in case of hardware failure). VM-to-PM affinities specify that a virtual machine should always be hosted on a particular physical machine, and thereby prohibit migrations of that virtual machine.

1) Virtual Machine Initial Placement: The initial placement of a virtual machine consists of two phases: first, the best (according to used objective function) physical machine to host a new virtual machine is identified (through running an optimization algorithm). The outcome of the first phase can also be a decision to not admit a virtual machine, if a data center is not able to provide required QoS (admission control). In such a case the requests potentially can be stored and reconsidered when enough computational resources are available, e.g., when a batch job is finished, after a scaling down action is performed, or when maintenance work is finished. In the second phase a new virtual machine is instantiated at that physical machine. Here we focus on the seconds phase and distinguish two typical cases of instantiation of a virtual machine: booting and cloning [14], [27].

In the first case, a virtual machine loads an operating system and application from an image saved in the storage. Depending on the operating system and the application, it can take even several minutes before the application achieves its full performance.

In case of cloning, the new virtual machine has to be a copy of an already running virtual machine. That case occurs when a placement is a result of a horizontal scaling. Live cloning of an already hosted virtual machine is a mean to shorten the time necessary to achieve full performance of the application in such a case. There are two main approaches to live cloning: post-copy and pre-copy (based on the same concepts as for virtual machine migration). A post-copy cloning [27], takes less than 1 second to create a replica and start processing on the new physical machine. However that 1 second does not include transfering memory, so after that time, still the source physical machine will experience the overhead of cloning. A pre-copy cloning [14], takes around 30 seconds to start the processing on the new physical machine, but after that time the source physical machine does not encounter any increased load due to cloning. Because of the differences in the time needed to start processing in the new location, the post-copy cloning is more suitable for handling unpredicted events. On the other hand, pre-copy seems to be more useful for events that were planned or predicted, because once the new instance is placed it does not consume any resources of source physical machine.

2) Virtual Machine Migration: Virtual Machine Migration is an action of moving an already hosted virtual machine from one physical machine (source) to another (target). It can cause an increase in power consumption and a degradation of QoS (e.g., response time) [3]. Two main types of virtual machine migration exist: live migration [28], which is suitable for services that should be available even during the time of migration, and cold migration [29] for applications that can be interrupted for the time of migration.

The time needed for applying a virtual machine migration depends mostly on the virtual machine's memory size, network bandwidth, and used migration technique. For live migration, similarly to VM cloning described in Section VI-C1, we distinguish post-copy and pre-copy approaches. The post-copy migration guarantees that the whole memory will be transfered just once. In case of a pre-copy migration some pages can be transfer several times if they got dirty after the first transfer and before the final migration of processing to the destination. Pre-copy is more applicable for proactive migrations, because it increases resource utilization on source physical machine during whole migration process. For reactive migrations, happening during unpredicted workload increases, post-copy seems to be a better choice, since it moves processing away from an overutilized physical machine immediately and later require only sending missing memory pages when they are needed.

3) vCPU-to-core Relationship: vCPU-to-core Relationship actions allow to change the number of physical cores assigned to virtual machine's virtual CPUs, as well as, to bind a virtual CPU to a particular physical core. Typically several virtual CPUs share one physical core in order to increase the resource utilization. To balance the load and counteract overheating of physical CPU cores, virtual CPUs are moved between different physical CPU cores inside a physical machine [30].

In case of multithread applications, the number of physical CPU cores assigned to the virtual machine has a direct influence on the performance. When the number of physical cores assigned to a virtual machine is smaller than the number of its virtual CPUs, the virtual CPUs are time-sharing the physical cores. If the CPU resource is the application's bottleneck, that may lead to a decreased performance. On the other hand, when the number of physical cores is higher that the number of virtual CPUs the virtual machine is not able to utilize all physical cores, and some of them stay idle. The optimal situation from the application's performance perspective is when the number of physical CPU cores is equal to the number of virtual CPUs and the virtual machine has an exclusive access to them. That contradicts the concept of sharing resources and increases the costs of hosting such application in the cloud. Therefore, the number of physical cores assigned to a virtual machine needs to be specified individually taking into account both the performance and costs.

CPU pinning may increase the performance of some applications by exploiting the temporal locality of references (data in the CPU cache) [31]. Thanks to running the application continuously on the same physical CPU core, a number of time-consuming and therefore performance-degrading events, e.g. cache misses, are avoided. On the other hand, CPU pinning may cause issues, e.g., limits schedulers ability to balance load across processors and interferes with fair sharing polices.

VII. Taxonomy of Sensors

Data center parameters are monitored to facilitate detection of changes that require immediate reaction, and creation of prediction models of workload and resource consumption. In this section, we describe the sensors used to monitor these parameters. First, we introduce a classification of monitored data according to the relative occurrence of their changes. Next, we describe individual parameters to be monitored.

TABLE V
Monitored Data Classification

Class	Examples	Monitoring
Static properties	Number of CPUs in PM	Triggered manually
Infrequently changing	VM-to-PM Mapping	Triggered on events
Frequently changing	CPU utilization	Constant

A. Relative Occurrence of Changes

We classify data that should be monitored into three classes according to the relative occurrence of changes of their values. Due to differences in the relative occurrence of changes, parameters require tailored ways of monitoring. Table V summarizes the differences between classes.

Static properties. There are some properties that are static (e.g., the total number of CPUs in a physical machine) and change only in case of physical interventions of an administrator, or in case of hardware failures. These properties do not have to be monitored constantly, but rather only when triggered by administrator after the initial setup or a manual change, or when an external system reports a failure. Static properties are used to update the data center configuration to ensure that management decisions are based on data that reflect introduced changes and detected failures.

Infrequently changing metrics. Infrequently changing metrics describe data that change as a result of applying management actions, e.g., a virtual machine placement or a change of physical machine state. Because of that, their update may be triggered when the management action is finished (or started in case of transient states like virtual machine migration).

Frequently changing metrics. Frequently changing metrics describe data that vary continuously, e.g., the utilization of resources at physical and virtual level, and the performance of applications. Because of frequent changes, metrics from this class have to be continuously monitored. Frequently changing metrics can be used for creating models describing the relationship between workload and resource utilization, for predicting future workloads, and for detecting spikes in workloads.

B. Monitored Data

Table VI shows possible data center sensors (data that can be monitored). We group data that needs to be monitored in three categories: physical layer, virtual layer, and application layer. There are many similarities between the configuration (actuators) and the monitored data (sensors), since they form a control feedback loop, described in this paper as a MAPE-K loop [7]. For every parameter type we specify related resources (if applicable), provide short description and identify the relative occurrence of changes.

1) Physical Layer: The current state of physical machines is monitored to give information on how much computational resources are currently available (i.e. running and able to host virtual machines) or can be available in longer time (e.g., in boot time for powered down physical machines, or in wake-up time for physical machines in power saving mode). Additionally, since physical machines can fail or loose network connection, such events should be noticed by a failure detection system and then recorded by monitoring system. Duration of transitions between states are measured to build a knowledge base useful for improving policies regarding powering physical machines down or putting them in a power saving mode. The frequency of CPUs is monitored and compared with the application behavior to find if there is still a possibility to improve the performance (by scaling frequency up) or decrease the power consumption (by scaling frequency down). All power management properties of physical machine listed in Table VI will normally change as a result of management actions, so the update of their values shall be triggered by respective PM Configuration actuators.

Physical machine specification parameters, describe the total amount of resources that a physical machine is equipped with. Despite the fact that the specification parameters are static, they still should be monitored because their values can change due to hardware failures (e.g., disk crash) or physical interventions of administrators (e.g., adding additional memory). Allocation of physical resources to virtual machines is monitored to keep track of available resources (i.e. resources not reserved for any virtual machine) and acceptable overbooking level. The value of allocation parameters change as an effect of VM Configuration actions and VM Migrations.

Utilization and performance parameters are constantly monitored to facilitate determining optimal resource utilization levels and detecting bottlenecks [32]. Power consumption shall be monitored to enable analysis of physical machine state changes and DVFS efficiency, and to capture the influence of resource utilization on power consumption. Collecting such data in the knowledge base and mining it can help finding the optimal configuration of frequency for given workloads and target resource utilization levels.

2) Virtual Layer: Autonomic manager keeps track of the changes in virtual machine's state which are infrequent and caused by management actions. VM State Changing actions can bring virtual machines into paused, suspended, or resuming state. VM Initial Placement actions cause initializing state, and VM Migration actions enter virtual machines into migrating state. Duration of virtual machine states shall be measured to build a knowledge base useful for predicting future changes and thus improving policies regarding migrating and suspending virtual machines. Since the duration may depend on the virtual machine's resource utilization, workload, and application performance, measurements of these parameters should be correlated.

Specifications of virtual machines need to be observed because they can change due to Vertical Scaling actions. Unsuccessful executions of management actions, e.g., releasing the available pages in memory ballooning [24] (deflating the balloon), may lead to differences between expected and actual configuration. Also hardware failure may cause a situation when a physical machine will not be able to provide previously allocated amount of resources.

Utilization at the virtual layer shows how much of resources allocated to a virtual machine are actually used by applications running inside and the guest operating system. Performance metrics of I/O operations can be used for bottleneck detection. Utilization and performance parameters at virtual layer are constantly monitored to model resource consumption and facilitate Vertical and Horizontal Scaling actions.

Placement of virtual machines is to be tracked to explore the dependencies between performance of virtual machines, and physical resource allocation and utilization, as well as, the effect of co-locating virtual machines on a same physical machine. Moreover, it is necessary to record the placement of virtual machines to be able to restore them in case of a physical machine crash. When manager founds that a physical machine is not reachable it instantiates copies of affected virtual machines on other physical machines to bring back the availability of services hosted on the corrupted machine.

3) Application Layer: Continuous monitoring of workload, e.g., the number of requests arriving to an application, is necessary to create a model of its changes over time. Information about the number of arriving requests can be combined with the resource utilization at the virtual machine level to form models of application behavior. These models can later be used for predictions of future workloads and resource consumption.

Monitoring throughput (the number of requests served by an application), response time (the time that is necessary to process requests), and application-specific KPIs (e.g., a number of transactions for a database), allows to identify if a proper QoS is provided. Comparing that information with virtual machine resource specification and utilization enables to optimize (minimize) the physical resource allocation while keeping satisfactory QoS and can be used to identify proper overbooking levels.

VIII. Sensor-Actuator Architecture

A. Hierarchy of Autonomic Elements

As presented in the Section V, there are many parameters that can be adjusted to optimize data center operations. Most of them can be changed independently for each physical or virtual machine, so the number of possible configurations in a production-size data center is enormous. Therefore, to facilitate effective use of available management actions, decision making processes and optimization can be distributed and organized in a hierarchical way. Autonomic elements at each level of hierarchy do not only directly control their own management actions, but also give guidelines for autonomic elements at the lower level. That indirect control helps to keep coherent management strategy and avoid conflicting actions.

In CACTOS [6] we use hierarchy of autonomic elements and assign to each level management actions, as follows.

Physical node. Management actions: DFVS, vertical scaling.

Cluster. Management actions: power up/down physical machines, power saving modes, VM initial placement and migration (on PM granularity), horizontal scaling.

Data center. Management actions: VM initial placement and migration (on cluster granularity).

B. Sensor-Actuator Interfaces

Since autonomic managers may operate at different levels, e.g., physical machine, cluster, whole data center, it is necessary to provide an efficient communication among sensors, actuators, and the rest of the system. Two interfaces shall be provided: the first, for collecting *monitored data* from sensors, and the second, for transferring the *optimization plans* to actuators. Figure 1 together with the steps of MAPE-K loop shows also the sensor-actuator interfaces.

TABLE VI
MONITORED DATA

Type	Resource	Description	RO
PHYSICAL LAYER			
Power Management	PM	current state	Infrequent
	PM	transition time between states	
	CPU	current frequency	
Specification	CPU	total number of CPUs and cores	Static
	Memory	total capacity	
	Storage	total capacity, read/write times	
	Network	nominal bandwidth, latency	
Allocation	CPU	number of allocated CPU cores	Infrequent
	Memory	total allocation	
	Storage	total allocation	
	Network	total allocation	
Utilization	CPU	utilization of each core	Frequent
	Memory	total utilization	
	Storage	total utilization	
	Network	total utilization	
Performance	Storage	measured read/write times	Frequent
	Network	measured bandwidth and latency	
Power	-	measured power consumption	Frequent
VIRTUAL LAYER			
State	VM	current state	Infrequent
	VM	transition time between states	
Specification	CPU	number of allocated physical cores	Infrequent
	Memory	allocated capacity	
	Storage	allocated capacity	
	Network	allocated bandwidth	
Utilization	CPU	utilization of each vCPU	Frequent
	Memory	utilization of allocated memory	
	Storage	utilization of allocated storage	
	Network	utilization of allocated bandwidth	
Performance	Storage	measured read/write times	Frequent
	Network	measured bandwidth and latency	
VM Placement	-	current placement of VMs	Infrequent
Pinning	CPU	current binding of vCPUs	Infrequent
APPLICATION LAYER			
Workload	-	number of requests arriving	Frequent
Throughput	-	number of requests served	
Response time	-	time of serving one request	
KPIs	-	application specific metrics	

RO – relative occurrence of changes in monitored data.

1) Monitored Data: Sensors need to be placed at each physical machine to constantly monitor selected parameters. Components responsible for Analyzing, Planning, and management of Knowledge, refereed here as an *optimizer*, may be separated from Sensors and running in a different location. Therefore, an interface describing monitored data is necessary to facilitate transferring data from sensors to the optimizer.

Measurements of physical machine's parameters and characteristics of hosted virtual machines have to be synchronized and consistent at the level of physical machines. It means that if a virtual machine runs on a physical machine that shall be reflected both in the physical machine's allocation and utilization, as well as, in VM-to-PM mapping and virtual machines utilization monitoring. Otherwise, it is impossible to drive conclusions about the relationships among application performance, virtual machine specification and physical machine resource utilization. However, synchronization and consistency across physical machines is not mandatory. Workload, resource utilization and performance models are based on the data from one physical machine. The only exception is for monitoring of actions in which more than one physical machine is involved, e.g., virtual machine cloning or migration.

2) Optimization Plan: Optimizers may be separated from actuators and run on dedicated physical machines to reduce the interference with other virtual machines. Because of that separation, it is necessary to provide an interface for transferring the output of optimizer, described in an optimization plan, to sensors. Optimization plans contain a list of management actions, e.g., migrate virtual machine x from physical machine y to physical machine z. Plans can contain a structure that describes relationships between management actions and indicate if they can be enacted in parallel or need to be processed in a serial manner, e.g., migrations to avoid network congestion. Since the execution of management actions can fail they shall be seen as a recommendations and not as commands that will be certainly implemented.

C. Reducing Monitoring Overhead

To achieve the objectives of monitoring (collecting data suitable for detection of unpredicted events and workload modeling), frequently changing workload, utilization, and performance parameters need to be monitored at high rate. However, monitoring of all data center entities (physical machines, virtual machines, and applications) at the same, high rate, results in a high performance overhead for network and computational resources, and huge amounts of data to be stored and analyzed. To reduce that overhead, monitoring systems should adjust the rate of monitoring automatically by: exploiting the knowledge about the pace of changes, differentiating between model creation and normal operation phases, and reacting on anomalies.

Similarly, the rate of monitoring should be adjusted to the pace of parameter's change. Not all data require periodic monitoring, e.g., static or infrequently changing parameters can be monitored only when management actions are executed or failures reported. Failure detection also requires some kind of probing, however the monitoring system can utilize information about detected failures instead of redundantly probing.

When creating a model, monitoring system should record changes of workload, utilization and performance at higher rate. However, during normal operation, it is not necessary to maintain the same monitoring rate. Instead, monitoring systems collect limited amount of parameters to verify if the model is precise enough to capture system behavior. When the model does not reflect reality anymore, the monitoring mode should be changed to model creation to update models.

When observing symptoms potentially preceding workload spikes or an unusual system behavior (parameter values out of standard range) monitoring system shall focus on these parameters and observe them with a higher rate. More data may allow to make more accurate decisions about the scale of workload spike or identify a false alarm.

IX. CONCLUSION

In this paper we address data center automation, and propose a sensor-actuator model for autonomic optimization of data center resource configuration based on an interpretation of the MAPE-K loop for autonomic systems. To construct a knowledge base for optimizers, the model characterizes different data center events, relates events to an identified set of actions that can be used to optimize data center operations (actuators), and discusses what data are needed for this type of optimization and at what timescales that data are available (sensors). The paper proposes taxonomies for selected aspects of data center configuration, elements of data center actuators, and classes of monitoring sensors. To support the analysis, the paper also provides characterizations and discussions of the relationships between data center events, sensors, and actuators; and presents results from a set of testbed experiments designed to illustrate selected trade-offs between these.

ACKNOWLEDGMENT

The authors would like to acknowledge Henning Groenda, Christian Stier, and Ahmed Ali-Eldin for constructive criticism and feedback on this work. This work is funded by the Swedish Research Council (VR) project Cloud Control, the Swedish Government's strategic research project eSSENCE, and the European Union's Seventh Framework Programme under grant agreement 610711 (CACTOS).

REFERENCES

[1] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," in *10th IEEE Int. Conference on High Performance Computing and Communications (HPCC '08)*, 2008, pp. 5–13.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Communication of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[3] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures," in *2010 IEEE 30th Int. Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2010, pp. 62–73.

[4] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware Cluster Management," in *19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*. ACM, 2014, pp. 127–144.

[5] C. Isci, J. Liu, B. Abali, J. Kephart, and J. Kouloheris, "Improving server utilization using fast virtual machine migration," *IBM Journal of Research and Development*, vol. 55, no. 6, pp. 4:1–4:12, 2011.

[6] P.-O. Östberg, H. Groenda, S. Wesner, J. Byrne, D. Nikolopoulos, C. Sheridan, J. Krzywda, A. Ali-Eldin, J. Tordsson, E. Elmroth, C. Stier, K. Krogmann, J. Domaschka, C. Hauser, P. Byrne, S. Svorobej, B. McCollum, Z. Papazachos, L. Johannessen, S. Rüth, and D. Paurevic, "The CACTOS Vision of Context-Aware Cloud Topology Optimization and Simulation," in *Proceedings of the Sixth IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2014.

[7] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[8] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[9] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically Scaling Applications in the Cloud," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, pp. 45–52, 2011.

[10] P. Svärd, B. Hudzia, S. Walsh, J. Tordsson, and E. Elmroth, "Principles and Performance Characteristics of Algorithms for Live VM Migration," *SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, pp. 142–155, 2015.

[11] B. Sotomayor, R. Montero, I. Martín Llorente, and I. Foster, "Resource Leasing and the Art of Suspending Virtual Machines," in *HPCC '09. 11th IEEE International Conference on High Performance Computing and Communications*, 2009, pp. 59–68.

[12] M. Maurer, I. Breskovic, V. C. Emeakaroha, and I. Brandic, "Revealing the MAPE loop for the autonomic management of Cloud infrastructures," in *2011 IEEE Symposium on Computers and Communications (ISCC)*, 2011, pp. 147–152.

[13] B. Jennings and R. Stadler, "Resource Management in Clouds: Survey and Research Challenges," *Journal of Network and Systems Management*, pp. 1–53, 2014.

[14] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: elastic distributed resource scaling for Infrastructure-as-a-Service," in *USENIX International Conference on Automated Computing (ICAC'13)*. USENIX, 2013, pp. 69–82.

[15] K. Morton, M. Balazinska, and D. Grossman, "ParaTimer: A Progress Indicator for MapReduce DAGs," in *2010 ACM SIGMOD International Conference on Management of Data*. ACM, 2010, pp. 507–518.

[16] A. Mehta, J. Dürango, J. Tordsson, and E. Elmroth, "Online Spike Detection in Cloud Workloads," in *IEEE International Conference on Cloud Engineering (IC2E 2015)*. IEEE, 2015, pp. 446–451.

[17] K. V. Vishwanath and N. Nagappan, "Characterizing Cloud Computing Hardware Reliability," in *1st ACM Symposium on Cloud Computing (SoCC '10)*. ACM, 2010, pp. 193–204.

[18] OW2 Consortium *et al.*, "RUBiS: Rice University Bidding System," *URL http://rubis.ow2.org*, 1999, accessed: 2015-05-26.

[19] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen, "GreenCloud: A New Architecture for Green Data Center," in *6th International Conference on Autonomic Computing and Communications Industry Session (ICAC-INDST '09)*. ACM, 2009, pp. 29–38.

[20] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *The Journal of Supercomputing*, vol. 60, no. 2, pp. 268–280, 2012.

[21] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Performance Evaluation*, vol. 67, no. 11, pp. 1155–1171, 2010.

[22] H. Qian and D. Medhi, "Server operational cost optimization for cloud computing service providers over a time horizon," in *11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*. USENIX, 2011, pp. 4–4.

[23] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *IEEE 14th International Symposium on High Performance Computer Architecture (HPCA 2008)*, 2008, pp. 123–134.

[24] C. A. Waldspurger, "Memory Resource Management in VMware ESX Server," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 181–194, 2002.

[25] S. Dutta, S. Gera, A. Verma, and B. Viswanathan, "Smartscale: Automatic application scaling in enterprise clouds," in *IEEE 5th Int. Conference on Cloud Computing (CLOUD '12)*, 2012, pp. 221–228.

[26] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth, "Modeling and Placement of Cloud Services with Internal Structure," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, to appear.

[27] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan, "SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing," in *4th ACM European Conf. on Computer Systems (EuroSys '09)*, 2009, pp. 1–12.

[28] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. USENIX, 2005, pp. 273–286.

[29] M. Kozuch and M. Satyanarayanan, "Internet suspend/resume," in *4th IEEE Workshop on Mobile Computing Systems and Applications*, 2002, pp. 40–46.

[30] I. Rodero, E. K. Lee, D. Pompili, M. Parashar, M. Gamell, and R. Figueiredo, "Towards energy-efficient reactive thermal management in instrumented datacenters," in *11th IEEE/ACM International Conference on Grid Computing (GRID)*, Oct 2010, pp. 321–328.

[31] T. Klug, M. Ott, J. Weidendorfer, and C. Trinitis, "autopin – Automated Optimization of Thread-to-Core Pinning on Multicore Systems," in *Transactions on High-Performance Embedded Architectures and Compilers III*, ser. Lecture Notes in Computer Science, P. Stenström, Ed. Springer Berlin Heidelberg, 2011, vol. 6590, pp. 219–235.

[32] O. Ibidunmoye, F. Hernandez-Rodriguez, and E. Elmroth, "Performance Anomaly Detection and Bottleneck Identification," *ACM Computing Surveys*, to appear.