A Classification of Middleware to Support Virtual Machines Adaptability in IaaS

José Simão
INESC-ID Lisboa
Instituto Superior de Engenharia de Lisboa
jsimao@cc.isel.ipl.pt

Luís Veiga
INESC-ID Lisboa
Instituto Superior Técnico
Universidade Técnica de Lisboa
luis.veiga@inesc-id.pt

ABSTRACT

The Infrastructure-as-a-Service (IaaS) model makes extensive use of virtualization to achieve workload isolation and efficient resource management. In general, the underlying supporting technologies are virtual machines monitors (e.g. hypervisors). Isolation is a static mechanism, relying on hardware or operating system support to be enforced. On the other hand, resource management is dynamic and middleware must be employed to adapt VMs in order to fit their guest's needs.

Although the services offered by virtual machines are used or extended in several works in the literature, the community lacks an organized and integrated perspective of the mechanisms and strategies regarding resource management and focusing on adaptation, which would allow for an effective comparison on the *quality* of the adaptation process.

In this work we review the main approaches for adaptation and monitoring in virtual machines deployments, their tradeoff, and their main mechanisms for resource management. We frame them into the control loop (monitoring, decision and actuation). Furthermore, we propose a classification framework that, when applied to a group of systems, can help visually in determining their similarities and differences.

Categories and Subject Descriptors

A.1 [**Introductory and Survey**]; D.4.1 [**Process Management**]: Scheduling; D.4.2 [**Storage Management**]: Virtual memory

Keywords

Virtualization, Resource-driven adaptability, Taxonomy

1. INTRODUCTION

Virtual machines (VM) are being used today both at the system and programming language level. At the system level they virtualize the hardware, giving the ability to guest multiple instances of an operating system on multi-core architectures, sharing computational resources in a secure way. At the high level programming languages, and similarly to the system level virtual machines, these VMs abstract from the underlying hardware resources, introducing a layer that can be used for fine grained resource control [13]. Furthermore, they promote portability through dynamic translation of an intermediate representation to a specific instruction set.

Virtual machines lay between guest systems and the underlying physical support (i.e. host hardware or operating systems) and are used to regulate resource usage, finding a way to partition the available resources by the guests. Most notably, VMs aim to optimize the operation of their guests, eventually using different algorithms or different parameters for each of them.

System level VMs, or hypervisors, are strongly motivated by the sharing of low-level resources. In result of this, many research and industry work can be found about how resources are to be delivered to each guest operating system. The partition is done with different reasonings, ranging from a simple *round robin* algorithm, to autonomic behavior where the hypervisor automatically distributes the available resources to the guests that, given the current workload, can make the best out of them. Among all resources, CPU [17, 6, 12] and memory [15, 8] are the two for which a larger body of work can be found. Nevertheless, other resources such as storage and network are also target of adaptation.

Virtual machines are not only a isomorphism between the guest system and a host [14], but a powerful software layer that can adapt its behavior, or be instructed to adapt, in order to transparently improve their guests's performance, minimizing the virtualization cost. In order to do so, VMs, and the middleware augmenting their services, can be framed into the well known adaptation loop [11]: i) monitoring or sensing, ii) control and decision, and iii) enforcement or actuation. Monitoring determines which components of the VM are observed. Control and decision take these observations and use them in some simple or complex strategy to decide what has to be changed. Enforcement deals with applying the decision to a given component/mechanism of the VM.

Adaptation is accomplished at different levels. As a consequence, monitoring, control and enforcement are applied in a way that have different impacts. For example, for the allocation of processing resources, the adaptation can be limited to the tuning of a parameter in the scheduling algorithm, the replacement of the algorithm, or the migration of the guest

ARM'12 December XX, 2012, Montreal, Quebec, Canada.

VM to another node.

In this work we present a framework to classify and help understand the quality of resource monitoring and adaptation techniques in virtual machines. Because of space restrictions, this paper focus on system level virtual machines, although the framework can also be applied to other virtualization layers such as high level virtual machines (i.e. managed runtimes). Section 2 presents the architecture of these VMs, depicting the building blocks that are used in research regarding resource usage and adaption. In Section 3 the classification framework is presented. For each of the resources considered, and for each of the three steps of the adaptation loop, we propose the use of a quantitative classification regarding the impact of the mechanisms used by each system. Furthermore, systems are globally classified regarding the dependency on low level monitoring, the complexity of control techniques and the latency of the enforcement mechanisms. We use this framework to classify state of the art systems in Section 5. Section 6 closes the document presenting some conclusions based on the previous discussion.

2. VIRTUAL MACHINES FUNDAMENTALS

Virtual machines have their roots in the 60's with the IBM 360 and 370 [2]. These systems provided a time-sharing environment where users had a complete abstraction of the underlying hardware resources. IBM goal was to provide better isolation among different users, providing *virtual machines* to each one. The architecture of the IBM System/370 was divided in three layers: the hardware, the control program (CP) and the conversational monitor system (CMS). The CP controlled the resource provision and the CMS delivered the services to the end user underpinned on these resources. The same architecture can be found in modern System VMs [3] where CP's role is given to the virtual machine monitor (VMM). Figure 1 depicts these three layers, where CP's role is given to the virtual machine monitor (VMM). The VMM purpose is to control the access of the guest operating systems running in each virtual machine to the physical resources, virtualizing processors, memory and I/O.

The next three sections will briefly describe how fundamental resources, CPU, memory and I/O are virtualized. The systems discussed in Section 5 are based on the building blocks presented here, extending them towards self-adaptation based on resource usage.

2.1 Computation as a resource

In a VM, virtualization of computation concerns two distinct aspects: *i)* the translation of instructions if the guest and host use a different Instruction Set Architecture (ISA) *ii)* the scheduling of virtual CPUs to a physical CPU (or CPU core on Symmetric Multiprocessors - SMP).

Instruction emulation (i.e. the translating from a set of instructions to another one) is common to both types of VMs. In System VMs, emulation is necessary to adapt different ISAs or in response to the execution of a privileged instruction (or a resource or behavior-sensitive instruction, even if not privileged) in the guest OS. Adaptation in binary, and byte code translation, is achieved by changing the translation technique (i.e. interpretation or compilation) and by replacing code previously translated with a more optimized one. These adaptations are driven by profiling information gather during program execution.

CPU scheduling is a well known issue in operating systems.A VMM scheduler has additional requisites when compared to the OS scheduler, namely the capacity to enforce a resource usage specified at the user's level. To achieve this, the CPU scheduler must take into account the *share* (or *weight*) given to each VM and make scheduling decisions proportional to this share [6]. However, in these schedulers, shares are not directly seen by the end user making it hard to define a high level resource management policy. Section 5 presents systems that dynamically change the scheduler parameters to give the VMM guests the resources that best fits their needs (e.g. performance, cost, energy consumption).

2.2 Memory as a resource

Memory is virtualized with a goal: give the illusion that guests have an virtually unbounded address space. Because memory is effectively limited, it will eventually end and the guest (operating systems or application) will have to deal with memory shortage. An extra level of indirection is added to the already virtualized environment of the guest operating systems. Operating systems give to their guests (i.e. processes) a dedicated address space, eventually bigger than the real available hardware.

The VMM can be managing multiple VMs, each with his guest OS. Therefore, the mapping between physical and real addresses must be extended because what is seen by an OS as a real address (i.e. machine address), can now change each time the VM hosting the OS is scheduled to run. The VMM introduces an extra level of indirection to the $virtual \rightarrow real$ mapping of each OS, keeping a $real \rightarrow physical$ to each of the running VMs.

When the VMM needs to free memory it has to decide which page(s) from which VM(s) to reclaim. This decision might have a poor performance impact. If the wrong choice is made, the guest OS will soon need to access the reclaimed page, resulting in wasted time. Another issue related to memory management in the VMM is the sharing of machine pages between different VMs. If these pages have code or read-only data they can be shared avoiding redundant copies. Section 5 present the way some relevant systems are built so that their choices are based on monitored parameters from the VM's memory utilization.

3. ADAPTATION TECHNIQUES USED IN IAAS SYSTEMS

In a software system, adaptation is regulated by monitoring, analyzing, deciding and acting [11]. Generically, the adaptability loop consists of three major steps: monitor, decision and action [11]. The monitor phase collects data from sensors. The decision phase determines what needs to be changed. Decisions made inside or outside the VM determine the complexity of the process. Finally, the action phase, applies the decision using the available effectors.

Adaptability mechanisms are not only confined to VM's internal structures but also to systems that externally reconfigure VM's parameters or algorithms. An example is the work of Shao et al. [12] to regulate VCPU to CPU mapping based on the CPU usage of specific applications.

The VMM has built in parameters to regulate how resources are shared by their different guests. These parameters regulate the allocation of resources to each VM and can be adapted at runtime to improve the behavior of the

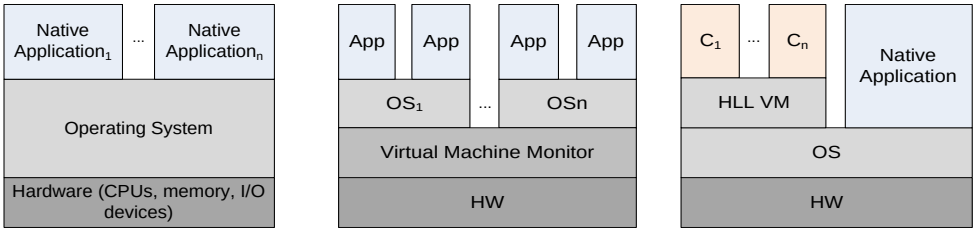

Figure 1: Virtualization layers

applications given a specific workload. The adaptation process can be internal, driven by profiling made exclusively inside of the VMM, or external, which depends on application's events such as the number of pending requests. In this section, the two major VMM subsystems, CPU scheduling and Memory Manager, will be framed into the adaptation processes.

3.1 CPU Management

An example of an exclusively inside activity is the CPU scheduling algorithm. To enforce the weight assigned to each VM, the hypervisor has to monitor the time of CPU assigned to each VCPUs of a VM, decide which VCPU(s) will run next, and assign it to a CPU [3, 6]. An example of an inside and outside management strategy is the one employed by systems that monitor events outside the hypervisor (e.g. operating systems load queue, application level events) [17, 12], use their own control strategy, such as linear optimization, control theory [10] or statistical methods [7]. Nevertheless, such systems act on mechanisms inside the hypervisor (e.g. weight assigned to VMs, number of VCPUs).

3.2 Memory Management

The memory manager virtualizes hardware pages and determines how they are mapped to each VM. To establish which and how many pages each VM is using, the VMM can monitor page's utilization using either whole page or sub-page scope. The monitoring activities aims to reveal how pages are being used by each VM and so information collected relates to *i)* page utilization [15, 16] and *ii)* page contents equality or similarity [15, 3]. Application performance (either by modification of the application or external monitoring) is also considered [8].

Because operating systems do not support dynamic changes to physical memory, the maximum amount of memory that can be allocated is statically assigned to each VM. Nevertheless, when total allocated memory exceeds the one that is physically available, the VMM must decide which clients must relinquish their allocated memory pages in favor of the current request. Decisions regarding memory pages allocation to each VM are made using *i)* shares [15], *ii)* history pattern [16] or *iii)* linear programming [8].

After deciding that a new configuration must be applied to a set of VMs, the VMM can enforce *i)* page sharing [15] or *ii)* page transfer between VMs. Page sharing relies on the mechanisms that exist at the VMM layer to map $real \rightarrow physical$ page numbers. On the other hand, the page transfer mechanism relies on the operating systems running at each VM, so that each operating system can use its own paging policy. This is accomplished using a balloon driver installed in each VM [3, 15].

3.3 Adaptation loop techniques

Figure 2 presents the techniques used in the adaptation loop. They are grouped into the two major adaptation targets, CPU and memory, and then into the three major phases of the adaptability loop.

4. CLASSIFICATION FRAMEWORK

To understand and compare different adaptation processes we now introduce a framework for classification of VM's adaptation techniques. It addresses the three classical adaptation steps. Each of this steps makes use of the different techniques described in the discussed in the previous section. We call it `RCI framework` because the analysis and classification of the techniques for each of these steps revolves around three fundamental aspects: *Responsiveness*, *Comprehensiveness* and *Intricateness*.

Responsiveness Represents how fast the system is able to adapt, thus it gets smaller as the following metrics increase: i) overhead of monitoring, ii) duration of the decision process, iii) the latency of applying adaptation actions.

Comprehensiveness Takes into account the breadth and scope of the adaptation process. In particular, it regards: i) the quantity or quality of the monitored sensors, ii) the totality of the elements considered for decision process, and iii) the number of different effectors that the system can engage.

Intricateness Addresses the depth of the adaption process. In particular, it regards low-level implications, interference and complexity of: i) the monitoring process, ii) decision strategy, and iii) enforcing actions.

These aspects were chosen, not only because they encompass many of the relevant goals and challenges in VM adaptability research, but mainly because they also embody a fundamental underlying tension: *that a given adaptation technique aiming at achieving improvements on two of these aspects, can only do so at the cost of the remaining one*. We came across this observation during the process of analyzing and classifying the techniques and systems studied.

Initially, we realized that no technique was able to combine full comprehensiveness and full intricateness, and still be able to perform without significant overhead and latency (possibly even requiring off-line processing). Later, we confirmed that full responsiveness always implies some level of restriction either to comprehensiveness or to intricateness. This *RCI conjecture* is yet another manifestation in systems research where the constant improvement on a given set of properties, or the behavior of a given set of mechanisms, can only come at an asymptotically increasing cost. This always forces designers to choose one of them to degrade in order

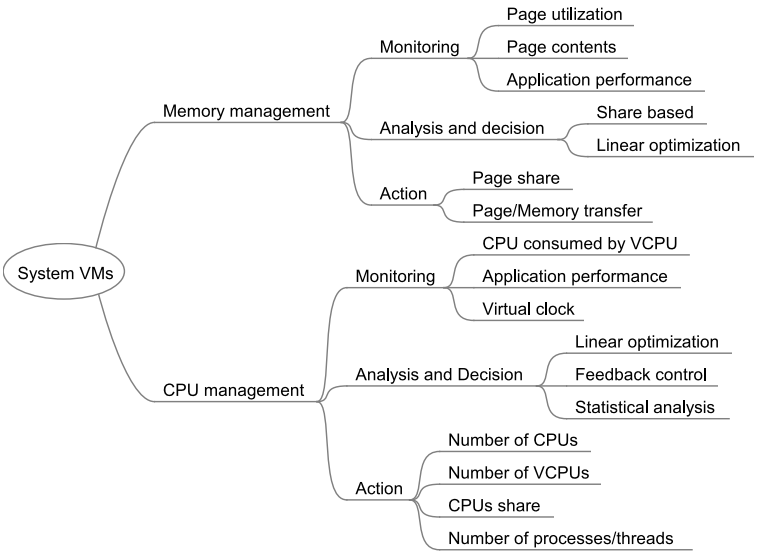

Figure 2: Techniques used by System VMs to monitor, control and enforce

to ensure the other two.

A paramount example is the CAP conjecture (or CAP theorem) [5], portraying the tension in large-scale distributed systems among (C)onsistency, (A)vailability, and tolerance to (P)artitions. Another example one is the tension, in the domain of peer-to-peer systems, among high availability, scalability, and support for dynamic populations [4].

Additionally, we also note that the tension inherent in the RCI conjecture is also present, at a higher-level of abstraction, among monitoring, decision, and action. The more the emphasis (regarded as an aggregate value of all RCI aspects) is given to two of the steps in the control loop, the less emphasis is possible to the remaining one, without breaking the viability and feasibility of the approach. We call this derived conjecture that applies to whole systems (and not to individual adaptation techniques) the MDA conjecture, for *Monitoring*, *Decision* and *Action*.

5. SYSTEMS AND THEIR CLASSIFICATION

In this section we briefly survey middleware used to provide resource usage adaptability for guests running on top of virtual machines. These systems rely on VMs capacity to adapt and are typically proposed to operate in multi-tenant infrastructures.

Xen [3]. In Xen each VM is called a *domain*. A special $domain_0$ (called *driver domain*) handles I/O requests of all other domains (called *guest domain*) and runs the administration tools. Because Xen's core solution is developed by the open source community, several works have studied Xen's scheduling strategies, for example in face of intensive I/O. Others propose adaptation strategies to be applied by the VMM regarding CPU to VCPU mapping or dynamically changing the scheduling algorithms parameters.

Xen includes three scheduling algorithms: Borrow Virtual Time (BVT), Simple Earliest Deadline First (SEDF) and Credit [1, 6]. The former two are deprecated and will probably be removed. Credit is a proportional fair scheduler. This means that the interval of time allocated for each VCPU is proportional to its *weight*, excluding small allocation errors. Additionally to *weight*, each *domain* has a *cap* value representing the percentage of extra CPU it can consume if his quantum has elapsed and there are idle CPUs. At each clock tick the running VPCUs are charged and eventually some will loose all their *credit* and tagged as *over* while the others are tagged *under*. VCPUs tagged as *under* have priority in scheduling decisions. Picking the next VCPU to run on a given CPU, Credit looks, in this order, a *under* VCPU from the local running queue, a *over* VCPU from the local running queue or a *under* VCPU from the running queue of a remote CPU, in a work-stealing inspired fashion.

Friendly Virtual Machines (FVM) [17]. The Friendly Virtual Machines (FVM) aims to enable efficient and fair usage of the underlying resources. Efficient in the sense that underlying system resources are nor overused or underused. Fairness in the sense that each VM gets a proportional share of the bottleneck resource. Each VM is responsible for adjusting its demand of the underlying resources, resulting in a distributed adaptation system.

The adaptation strategy is done using feedback control rules such as Additive-Increase and Multiplicative-Decrease (AIMD), driven by a single control signal - the *Virtual Clock Time* (VCT) to detect overload situation. VCT is the real time taken by the VMM to increment the virtual clock of a given VM. An increase in VCT means that the host VMM is taking longer to respond to the VM which indicates a contention on a bottlenecked resource. Depending on the nature of the resource the VCT will evolve differently as more VMs are added to the system. For example, with more VMs sharing the same memory, more page faults will occur, and even a small increase in the number of page faults will result in a significant increase in VCT.

A VM runs inside a hosted virtual machine, the User Mode Linux, an so, two types of mechanisms are used to adapt VM's demand to the available underlying resources. FVM imposes upper bounds on *i)* the Multi Programming Level (MPL) and on *ii)* the rate of execution. MPL controls the

number of processes and threads that are effectively running at each VM. When only a single thread of execution exists, FVM will adapt the rate of execution forcing the VM to periodically sleep.

HPC computing [12]. Shao et al. adapts the VCPU mapping of Xen [3] based on runtime information collected by a monitor that must be running inside each guest's operating system. They adjust the numbers of VCPUs to meet the real needs of each guest. Decisions are made based on two metrics: the average VCPU utilization rate and the parallel level. The parallel level mainly depends on the length of each VCPU's run queue. The adaptation process uses an addictive increase and subtractive decrease (AISD) strategy. Shao et al. focus their work on native applications representative of high performance computing applications.

Ginko [8]. Ginko is an application-driven memory overcommitment framework which allows cloud providers to run more System VMs with the same memory. For each VM, Ginkgo uses a profiling phase where it collects samples of the application performance, memory usage, and submitted load. Then, in production phase, instead of assigning the same amount of memory for each VM, Ginko takes the previously built model and, using a linear program, determines the VM ideal amount of memory to avoid violations of service level agreements. This means that the linear program will determine the memory allocation that, for the current load, maximizes the application performance (e.g. response time, throughput).

Auto Control [10]. Padala et al. proposes a system which uses a control theory model to regulate resource allocation, based on multiple inputs and driving multiple outputs. Inputs are applications running in a VMM and can spawn several nodes of the data center (i.e. web and DB tier can be located in different nodes). Outputs are the resource allocation of CPU and disk I/O caps. For each application, there is an application controller which collects the application performance metrics (e.g. application throughput or average response time) and, based on the application's performance target, determines the new requested allocation. Because computational systems are non linear, the model is adjusted automatically, aiming to adapt to different operating points and workloads. Based on each application controller output, a per node controller will determine the actual resource allocation. It does so by solving the optimization problem of minimizing the penalty function for not meeting the performance targets of the applications. To evaluate their system, applications were instrumented to collect performance statistics. Xen monitoring tool (i.e. `xm`) was used to collect CPU usage and `iostat` was used to collect CPU and disk usage statistics. Enforcement is made by changing Xen's credit scheduler parameters and a proportional-share I/O scheduler.

PRESS [7]. PRESS is an online resource demand prediction system, which aims to handle both cyclic and non-cyclic workloads. It tracks resource usage and predicts how resource demands will evolve in the near future. To detect repeating patterns it employs signal processing techniques (i.e. Fast Fourier Transform and the Pearson correlation), looking for a signature in the resource usage history. If a signature is not found PRESS uses a discrete-time Markov chain. This technique allows PRESS to calculate how the system should change the resource allocation policy, by transiting to the highest probability state, given the current state. In [7] the authors focus on CPU usage. So, The prediction scheme is used to set the CPU cap of the target VM. The evaluation was made based on a synthetic workload applied to the RUBiS benchmark, built from observations of two real world workloads.

VM³ [9]. The work in VM³ aims at measuring, modelling and managing shared resources in virtual machines. It operates in the context of virtual machine consolidation in cloud scenarios proposing a benchmark (vConsolidate). It places emphasis on balancing quickness of adaptation and the intricateness and low-level of the resources monitored, while sacrificing comprehensiveness by being restricted to deciding migration of virtual machines among cluster nodes.

5.1 Analysis

In Figure 3 we analyze the *responsiveness*, *comprehensiveness* and *intricateness* aspects for the different adaption techniques used in System VMs, and in the context of the adaptation steps *monitoring*, *decision* and *action* in System VMs. In the former, to find a system's RCI, we assign a R, C and I to each technique and sum the values regarding each technique used by the system. The later, is a second order analysis in the sense that the value for M, D and A of a system is found by averaging the RCI values of each of his techniques.

These results allows us to make four major conclusions. First, different systems have a different RCI coverage. Second, intricateness seems to dominate although responsiveness is also high in most systems. Third, systems with larger responsiveness and intricateness are less comprehensive. Finally, decision strategies and effectors are similar while monitoring techniques are more heterogeneous.

6. CONCLUSIONS

In this work we reviewed the main approaches for adaptation and monitoring in virtual machines, their tradeoffs, and their main mechanisms for resource management. We frame them into the control loop (monitoring, decision and actuation). Furthermore, we proposed a novel taxonomy and classification framework that, when applied to a group of systems, can help visually in determining their similarities and differences. Supported by this, we presented a comprehensive survey and analysis of relevant techniques and systems in the context of virtual machine monitoring and adaptability.

This taxonomy was inspired by two conjectures that arise from the analysis of existing relevant work in monitoring and adaptability of virtual machines. We presented the RCI conjecture on monitoring and adaptability in systems, identifying the fundamental tension among Responsiveness, Comprehensiveness, and Intricateness, and how a given adaptation technique aiming at achieving improvements on two of these aspects, can only do so at the cost of the remaining one. Then we presented a derived conjecture, the MDA conjecture identifying a related tension, in the context of whole systems, among emphasis on monitoring, decision and action. Regarding future work, we plan to apply the frame-

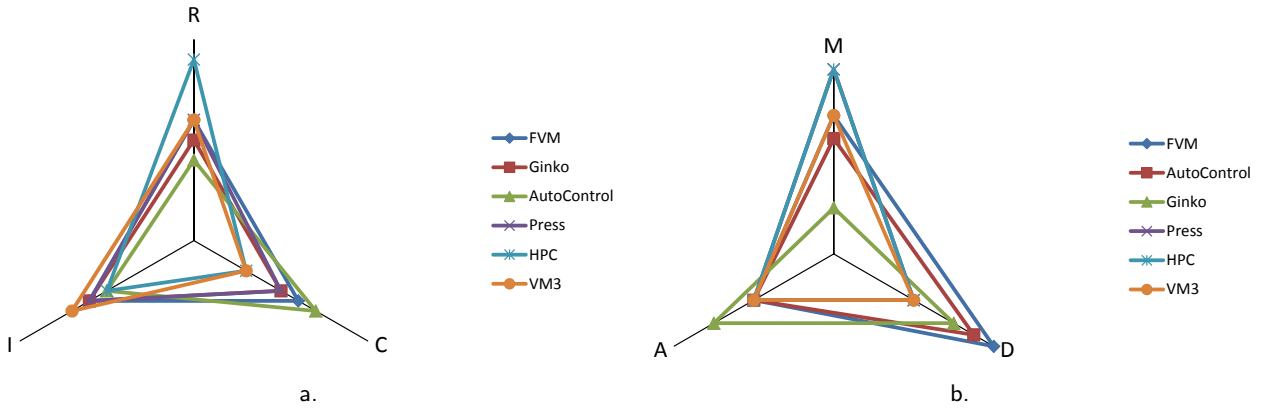

Figure 3: Relationship of emphasis on *responsiveness*, *comprehensiveness* and *intricateness* regarding a. adaption techniques and b. the adaptation steps *monitoring*, *decision* and *action* in System VMs.

work to other virtualization layers, such as high level virtual machines (i.e. managed runtimes), adding new systems to the analyses and new adaptation techniques.

Acknowlegments. This work was partially supported by national funds through FCT - Fundação para a Ciência e a Tecnologia, under projects PTDC/EIA-EIA/102250/2008, PTDC/EIA-EIA/108963/2008, PTDC/EIA-EIA/113613/2009, and PEst-OE/EEI/LA0021/2011

7. REFERENCES

[1] http://wiki.xen.org/wiki/credit_scheduler, visited at 3-10-2012.

[2] G. M. Amdahl, G. A. Blaauw, and F. P. Brooks. Architecture of the ibm system/360. *IBM J. Res. Dev.*, 8:87–101, April 1964.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37:164–177, October 2003.

[4] Charles Blake and Rodrigo Rodrigues. High availability, scalable storage, dynamic peer networks: Pick two. In Michael B. Jones, editor, *HotOS*, pages 1–6. USENIX, 2003.

[5] Eric A. Brewer. A certain freedom: thoughts on the cap theorem. In Andréa W. Richa and Rachid Guerraoui, editors, *PODC*, page 335. ACM, 2010.

[6] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev.*, 35:42–51, September 2007.

[7] Zhenhuan Gong, Xiaohui Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9 –16, oct. 2010.

[8] Michael Hines, Abel Gordon, Marcio Silva, Dilma Da Silva, Kyung Dong Ryu, and Muli Ben-Yehuda. Applications know best: Performance-driven memory overcommit with ginkgo. In *CloudCom '11: 3rd IEEE International Conference on Cloud Computing Technology and Science*, 2011.

[9] Ravi Iyer, Ramesh Illikkal, Omesh Tickoo, Li Zhao, Padma Apparao, and Don Newell. Vm3: Measuring, modeling and managing vm shared resources. *Computer Networks*, 53(17):2873–2887, December 2009.

[10] Pradeep Padala, Kai-Yuan Hou, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, and Arif Merchant. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 13–26, New York, NY, USA, 2009. ACM.

[11] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4:14:1–14:42, May 2009.

[12] Zhiyuan Shao, Hai Jin, and Yong Li. Virtual machine resource management for high performance computing applications. *Parallel and Distributed Processing with Applications, International Symposium on*, 0:137–144, 2009.

[13] José Simão, João Lemos, and Luís Veiga. A^2-VM : A cooperative Java VM with support for resource-awareness and cluster-wide thread scheduling. In *OTM Conferences*, volume 7044 of *Lecture Notes in Computer Science*, pages 302–320. Springer, 2011.

[14] Jim Smith and Ravi Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann, 2005.

[15] Carl A. Waldspurger. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, 36:181–194, December 2002.

[16] Zhao Weiming and Wang Zhenlin. Dynamic memory balancing for virtual machines. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 21–30, 2009.

[17] Yuting Zhang, Azer Bestavros, Mina Guirguis, Ibrahim Matta, and Richard West. Friendly virtual machines: leveraging a feedback-control model for application adaptation. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, VEE '05, pages 2–12, New York, NY, USA, 2005. ACM.