In: Proceedings of the 15th Annual Workshop of HP Software University Association, Hosted by Al Akhawayn University in Ifrane, June 22 - 25, 2008

A Framework for Autonomic Performance Management of Virtual Machine-Based Services

Markus Schmid, Dan Marinescu, and Reinhold Kroeger

Wiesbaden University of Applied Sciences
Distributed Systems Lab
Kurt-Schumacher-Ring 18, D-65197 Wiesbaden, Germany
{schmid,dan.marinescu,kroeger}@informatik.fh-wiesbaden.de

Abstract. This paper presents the design for a modular framework for autonomic QoS management of virtual machine environments. The framework separates the management APIs provided by individual virtual machine technologies from the high-level control algorithms, which makes the design of generic controllers possible. In addition, control algorithms can be replaced without effort, which provides an easy way for comparing the performance of different control algorithms. We discuss first approaches to control algorithms that were designed the framework. In addition, we present the prototypical implementation and show a number of test results that were gained during the validation of the approach. The paper closes with a conclusion and an overview on future work.

1 Motivation

System virtualisation is a technique that was first developed in the mid 1960's. It introduces a layer of indirection between hardware and operating system, called *virtual machine monitor* (VMM). The VMM provides support for creating and running multiple virtual machines (VMs) in parallel that share the underlying hardware. Originally, virtualisation techniques were developed to increase the overall system reliability by isolating individual applications from each other [12].

Over the past years, system virtualisation became a hot topic again, being on the rise in data centres as well as on desktop systems [16, 19, 15]. The adoption of virtualisation in data centres is taking place at a high pace, as it allows reducing hardware and maintenance costs by consolidating numerous, mostly under-utilised servers. Virtualisation may reduce system downtimes caused by hardware defects, as VMs can be seamlessly migrated in between different physical hosts without need for a system shutdown [2, 3].

However, in general, virtualisation significantly increases the overall complexity of computing systems as administrators have to deal with a potentially much bigger number of (virtual) computer systems that still require adequate monitoring and management. In a virtualised data centre, the overall number of inter-component dependencies increases tremendously and wrong management decisions can potentially cause huge damage. At the same time, virtualised data

In: Proceedings of the 15th Annual Workshop of HP Software University Association, Hosted by Al Akhawayn University in Ifrane, June 22 - 25, 2008

centres offer a huge potential for optimisation, both in the reduction of physical resource allocation and power consumption ("Green-IT") and the possibility to offer a great range of Quality of Service (QoS) classes for provided services.

In the long run, it will not be possible to efficiently handle the management of virtualised data centres without significant automation and the development of self-management approaches [8] that are capable of handling standard management tasks without human interaction and support for QoS-oriented resource allocation. Automated common management for VM environments from different vendors is currently aggravated by the diversity of existing, mainly vendor-specific control interfaces and tools.

This paper presents a modular framework for autonomic QoS management of VM-based application services. The framework separates the management APIs provided by individual VMM technologies from the high-level controllers used for autonomic management, which makes the design of VMM-independent controllers possible.

The paper is structured as follows: In section 2 we discuss our modular self-management architecture for VMs, while section 3 shows first approaches to control algorithms that rely on our management architecture. Section 4 presents the prototypical implementation of this approach in combination with measurements and first practical experiences. Section 5 describes related work to the automation of management in VM environments and briefly compares the presented approaches to our work. Section 6 concludes the paper and gives an overview on future enhancements.

2 The Autonomic Management Framework

We have designed a management framework for VM environments in virtualised data centres. The framework supports different types of intelligent controllers by providing a generic controller interface. From the controller's perspective this acts as an abstraction layer on top of the virtualisation technology.

The framework is designed to manage a pool of n physical machines, each hosting between 0 and m VMs. It comprises three types of management components: *VM Managers* are responsible for one VM each, one *PM Manager* is assigned to each physical machine (PM) within a pool and a *Pool Manager* takes care of the entire pool (see fig. 1). All manager components comprise sensor and actuator modules that can easily be adapted to different vendor-specific VM control interfaces. In addition, VM Managers and Pool Managers contain *logic modules* which host the architecture's autonomic control algorithms.

The VM Manager component monitors and controls both, the OS and the applications of a VM (in the following termed services). It monitors OS parameters like CPU utilisation and used/available memory as well as specific QoS parameters regarding the service hosted by the VM. We assume that each VM hosts only a single service, e.g. Web server, mail server or DBMS. We argue that this is common practise in a server consolidation scenario. Furthermore, we define service level objectives (SLOs) for the services provided by the VMs. The

In: Proceedings of the 15th Annual Workshop of HP Software University Association, Hosted by Al Akhawayn University in Ifrane, June 22 - 25, 2008

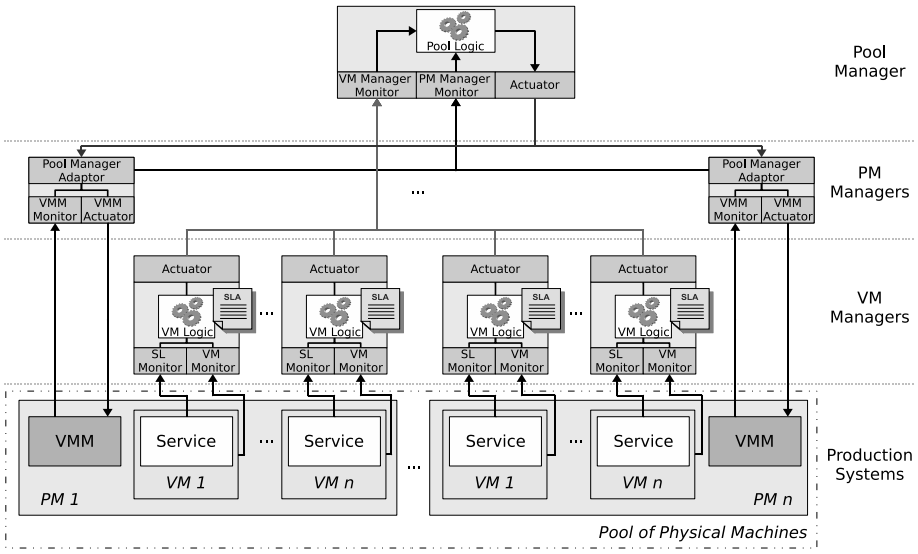

Fig. 1. Overall architecture of the autonomic VM management framework

VM Logic module uses the gathered monitoring information to detect SLO violations and aims at determining the resource bottleneck (e.g. CPU or memory) that causes the problem. After having identified a bottleneck, the VM Manager informs the responsible Pool Manager and requests an appropriate reaction.

The PM Manager component controls a physical machine, basically by executing two different tasks: it observes the resource utilisation of the machine and forwards the collected data to the Pool Manager. In addition, the PM Manager executes management actions that are requested by the Pool Manager through its Actuator module, e.g. resource reallocation or migration of VMs to a different physical machine based on the migration functions of the underlying virtual machine monitor (VMM).

The Pool Manager component comprises a Monitor module which receives monitoring data from the PM Managers within the pool. It uses the aggregated data to create a global view of resource usage and availability in the managed pool. This global view is used by the *Pool Logic* module to fulfil (VM Manager) requests for additional resources. Having determined a way to reallocate resources for VMs in the pool, the Pool Manager uses its Actuator component to request PM Managers to perform the required resource allocation changes.

2.1 Internal Architecture of a Manager Component

In compliance with the IBM autonomic computing reference architecture [17], we have developed a modular *Self-Manager* that provides a customisable basis

In: Proceedings of the 15th Annual Workshop of HP Software University Association, Hosted by Al Akhawayn University in Ifrane, June 22 - 25, 2008

for the VM Managers, the PM Managers and the Pool Manager (see fig. 2 for details).

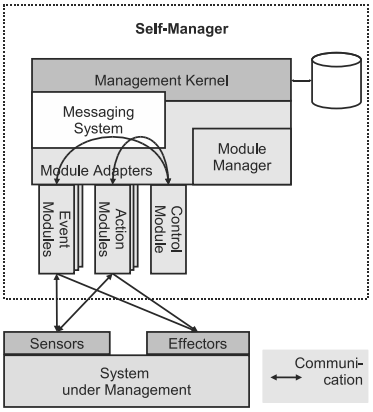

Fig. 2. Modular Self-Manager architecture

Internally, the Self-Manager consists of four main components: The central element of the Self-Manager is the `management kernel`, which provides `adapters` to connect extension modules. Extension modules are instantiated by the central `module manager`. The modules implement sensors, effectors, and the internal logic of the self-management controller. A `messaging subsystem`, which is part of the kernel, is responsible for message handling within the Self-Manager.

The Self-Manager supports three kinds of extension modules: `event modules`, `action modules`, and `control modules`. `Event modules` create their own threads and thus are able to react actively to changes within the environment, e.g. by creating internal messages. `Action modules` are passive; they act – triggered by internal messages – by analysing application-specific sensors, or performing management tasks. Sensors can be realised using either `event modules` (push model) or `action modules` (pull model). Application-specific actuators are realised through `action modules`. The modules implement application-class specific interfaces, e.g. for accessing the control API of a VMM.

`Control modules` form the "brain" of the Self-Manager as they host the management knowledge and implement the control algorithms. `Control modules` act periodically or are triggered by incoming messages. Management decisions are communicated to other modules using the internal messaging capabilities.

Internally, VM Managers and the Pool Manager comprise a control module and several action and event modules, while the PM Managers solely consist of action and event modules. A number of controller modules are presented in the following section. The architecture uses Java RMI-based event modules for inter-manager communication.

The management framework can be easily adapted to different virtualisation technologies by simply replacing the technology-dependent action and event modules in the VM Managers and the PM Managers.

3 Examples for Autonomic Control Algorithms

To demonstrate the flexibility and adaptability of our management framework we implemented first examples of self-management control approaches for both, the VM Logic and Pool Logic components. All approaches presented in the following

In: Proceedings of the 15th Annual Workshop of HP Software University Association, Hosted by Al Akhawayn University in Ifrane, June 22 - 25, 2008

use *service response time* as example SLA parameter for service monitoring. The use of other SLA parameters (e.g. throughput) is however possible.

3.1 Autonomic Management for a VM Logic Component

The purpose of a VM Logic component is to monitor and analyse the resource utilisation of a VM and the response times of the service hosted by the VM. Once an SLO violation occurs, the VM Logic component uses the data representing the resource utilisation of the VM to determine the root cause of this violation. Thus, the VM Logic component needs to perform a bottleneck determination process, just like a human system administrator would do. This is a multi-dimensional problem as resources have interdependencies. In a first approach we concentrated on a single resource (memory) and modelled the bottleneck-determination process as a rule-based expert system. The controllers' knowledge about the managed system is represented as a number of rules, such as:

facts : SLO violation occurred, resource X below threshold
action : request increase in resource X

At runtime, the rule engine uses input data like service response times and VM resource utilisation to try to match the facts of a rule. If all facts of a rule match, the corresponding action is performed. For the given example, the VM Logic Component sends a request for an increase in resource X to the Pool Manager, which then evaluates all incoming requests and takes appropriate actions from its global perspective.

3.2 Autonomic Management for the Pool Logic component

Due to the abstraction layer provided by the VM Managers and the PM Managers, the Pool Manager solely deals with the problem of satisfying the resource requests of the different VM Managers. Thus, at this abstraction level, we see resource consumers (the VMs), and resource providers (the physical machines). We thus aim to satisfy the needs of the resource consumers using the resources supplied by the resource providers.

Various approaches can be used to solve this resource allocation problem. We present two different heuristic-based algorithms that try to solve our resource allocation problem, currently only considering one resource type at a time. However, further research in appropriate control algorithms is needed as in general the Pool Logic components are faced with a NP-hard multi-dimensional multiple knapsack problem, as discussed in [9].

Algorithm A first checks whether the additional resource share required by a VM is available on the physical machine that currently provisions the VM. If this is the case, the algorithm gives instructions to allocate the additional resource share to the VM and terminates. If the requested resource share is not available on the current physical machine, the algorithm looks for a different physical machine

In: Proceedings of the 15th Annual Workshop of HP Software University Association, Hosted by Al Akhawayn University in Ifrane, June 22 - 25, 2008

in the pool that provides enough spare resources for hosting the requesting VM. This means that not only the requested resource share must be available, but in addition all the resources currently assigned to the VM. If such a remote physical machine is found, the VM is migrated to its new destination and the algorithm terminates. The selected destination machine can be either online or in an offline state. However, destinations that are currently offline cause higher migration costs, due to the time span needed for powering up and booting. In case no suitable destination is found, the algorithm fails to find a solution.

This algorithm uses a "first fit" strategy. The disadvantages of Algorithm A are obvious: besides not always being capable of finding a solution, the solutions found by the algorithm can sometimes be quite far from optimal. On the other hand, the algorithm has a linear complexity. The solutions found the by Algorithm A, although not always optimal, represent states that can be achieved by performing a single migration from the initial state. This ensures that no costly solution will ever be suggested by the algorithm.

Algorithm B uses a local search approach based on the K-Best-First Search (KBFS) algorithm, first introduced by Felner et al. [5]. Our algorithm encodes the mapping of VMs to physical machines in the pool in form of states. In addition, the encoding includes allocated and available resources of each physical machine. Each state has its global profit, representing a function of the sum of the local profits of each physical machine and the validity of the state.

The local profit of a physical machine represents a function of its resource utilisation. Thus, the higher the resource utilisation of a machine, the higher the local profit. An offline physical machine has the maximum local profit possible. This assures that the global profit of a state increases when some machines are kept at a high resource utilisation while the others are kept offline.

Global profit is calculated by multiplying the sum of the local profits with the validity of the state, which can be either `0` or `1`. Thus, the global profit can either be the sum of the local profits if the state is valid, or `0` if the state is not valid. A valid state is a state where the free capacity (e.g. in terms of memory, CPU) of any physical machine is bigger or equal to the sum of requested capacities of the VMs provisioned on that physical machine. All other states are invalid. As a result, Algorithm B defines a valid global state as a state with a global profit higher than `0`. However, valid states can also be ranked, depending on their global profit.

Besides the global profit of a state, Algorithm B also computes the total cost of moving from the initial state to any given state. The total cost is a function of the number of migrations necessary for the translation from the initial state to that given state. The total cost is then used in combination with the global profit to determine the utility of a translation between an initial state and a given state. This is the ultimate quality factor that is used to select the final state.

Using this encoding and criteria, Algorithm B performs its local search. For this, two lists are used: the *open list* of states (nodes) that have been evaluated with respect to their utility but not expanded, and the *closed list* which contains

In: Proceedings of the 15th Annual Workshop of HP Software University Association, Hosted by Al Akhawayn University in Ifrane, June 22 - 25, 2008

the nodes that have already been expanded. At each iteration, the best k nodes from the open list are expanded; their children are evaluated and added to the open list. After a limited number of iterations, the algorithm terminates and the node with the highest utility is selected. This represents the final state, and the system moves to that state by means of successive migrations.

Algorithm B is always capable to find a valid solution if valid solutions exist and the translation costs to at least one of these solutions does not exceed a certain, predefined value. However, in finding a solution the algorithm determines the local optimum which is not necessarily the global optimal solution.

4 Implementation and First Experiences

We have implemented a Java-based prototype of the management framework. The prototype is able to autonomically manage VMs based on the Xen hypervisor. The implementation comprises sensors and effectors to access the Xen API [18] for VM reconfiguration and migration.

The implemented VM Managers use JMX [7] interfaces to monitor VM parameters and service response time metrics.

The PM Managers use the Xen-API to manage a physical machine that hosts an instance of the *Xen 3.1* hypervisor. The Xen-API is used for monitoring, dynamic resource allocation, and to perform live VM migration. Both the VM Managers and the PM Managers communicate with the Pool Manager using Java RMI. The Pool Manager provides support for plugging-in different Pool Logic components through a messaging interface.

Examples for VM Logic and Pool Logic components have been implemented as presented in section 3. We used the rule engine *Jess* in the implementation of the VM Logic approach. Our prototype uses *maximum service response time* as example SLA parameter for service monitoring. At runtime, the rule engine uses the monitored service response times and VM memory and swap space utilisation to try to match all facts of a rule. On success, the corresponding action (e.g. reconfiguration or notification of the Pool Manager) is performed.

To determine the rules for the VM manager's VM Logic, we manually observed the impact of memory allocation changes to the overall service behaviour. In principle this process can be at least partially automated for a service class as shown in [4]. Fig. 3 gives an example for the dependency of service response times on the amount of memory assigned to a VM. The figure depicts the overall response times measured for the TPC-W benchmark [14] servlet `new_products`. We restarted the physical machine in between the test runs. Therefore we can observe response time peaks at the start of each test run, which are caused by the necessary initialisation of the underlying tomcat server (e.g. application class loading) after server start up and do not occur after the initial warm up period. Based on these observations we defined a number of rules, such as:

facts : Response time SLO violation occurred, the OS is swapping
action : Request allocation of additional memory

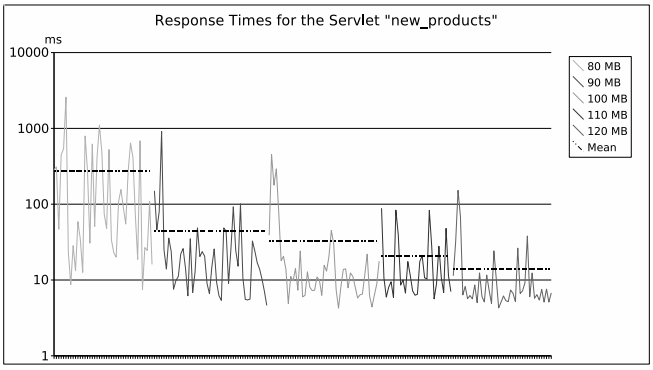

Fig. 3. Response times depend on the amount of main memory assigned

The framework and the implemented autonomic control algorithms have been successfully tested in a lab environment, using the TPC-W benchmark as an example application service.

The TPC-W e-commerce application suite is provisioned by a Tomcat server that runs in a Xen-based VM. The associated VM Manager enforces an SLO, which defines a maximum response time of 3000ms, tolerating a maximum of three violations within a two minute period. As a TPC-W load generator is started on a client system, the VM Manager observes a dramatic increase in response times, resulting in an SLO violation. Thus, the VM Manager requests additional memory from the Pool Manager. As this request is fulfilled, the VM Manager observes a significant decrease in response times down to an acceptable level. This behaviour can be observed in fig. 4. The decrease in SLO violations per minute shows that the VM Manager is able to successfully determine the resource that caused the bottleneck.

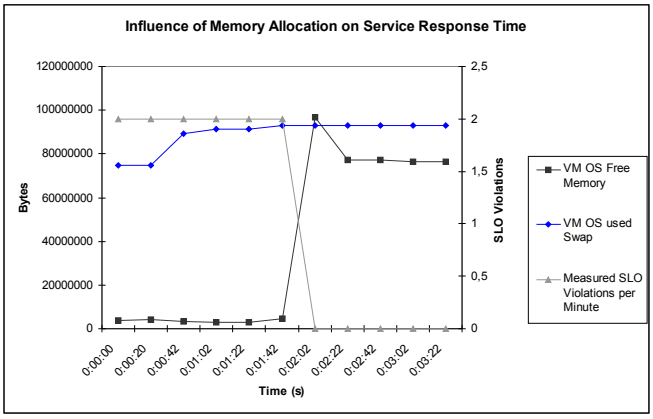

Fig. 4. Influence of available memory on the number of SLO violations

In: Proceedings of the 15th Annual Workshop of HP Software University Association, Hosted by Al Akhawayn University in Ifrane, June 22 - 25, 2008

The two Pool Logic algorithms described in section 3 have been evaluated live and by means of simulation. Since in the previous test the VM Manager has determined memory to be the bottleneck resource, we concentrated on memory allocation. In the following we describe our experiences in two scenarios.

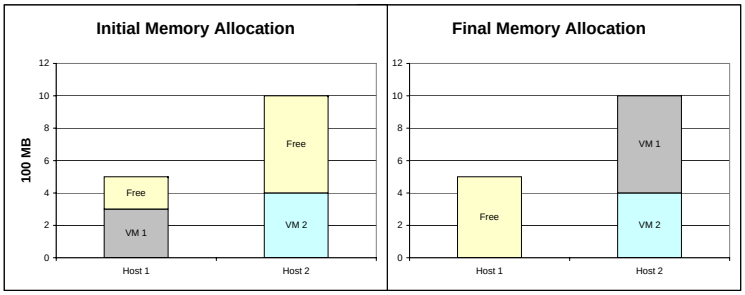

Fig. 5. A management scenario comprising two physical machines

In the first scenario, two physical machines named Host 1 and Host 2 each provision one VM, VM 1 respectively VM 2 (see fig. 5). Host 1 possesses a total of 500 MB RAM for VMs, while Host 2 provides 1000 MB. At the beginning, VM 1 uses 300 MB, while VM 2 uses 400 MB. As the TPC-W load generator stresses VM 1, the VM Manager responsible for VM 1 requests additional memory (in this case an extra 300 MB) from the Pool Manager. In this scenario, both Pool Logic algorithms deliver the same solution, namely to migrate VM 1 from Host 1 to Host 2 as Host 1 is not able to provide extra 300 MB of memory.

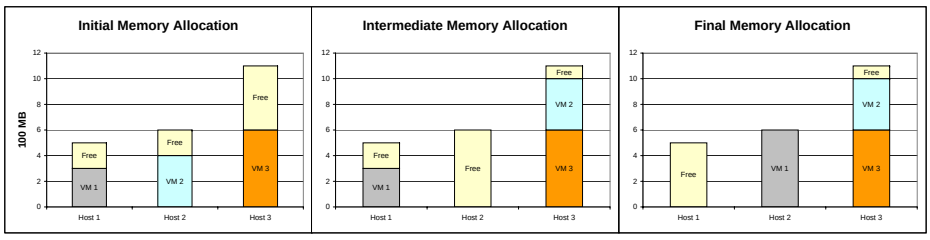

Fig. 6. A management scenario comprising three physical machines

The second scenario comprises three physical machines: Host 1, Host 2 and Host 3 (see fig. 6, left chart). At the beginning, each of them provisions one VM: VM 1, VM 2 and VM 3 respectively. In this scenario, Host 1 possesses a total of 500 MB RAM for VMs, Host 2 600 MB and Host 3 1100 MB, while VM 1 uses 300 MB, VM 2 400 MB and VM 3 600 MB. Again, the VM Manager responsible for VM 1 requests 300 MB of additional memory. Unlike in the first scenario, the

In: Proceedings of the 15th Annual Workshop of HP Software University Association, Hosted by Al Akhawayn University in Ifrane, June 22 - 25, 2008

two algorithms perform differently. In fact, Algorithm A is not even capable of finding a valid solution as neither Host 2 nor Host 3 has 600 MB of free memory available. This scenario clearly shows the limitations of this simple algorithm. The more sophisticated Algorithm B is able to find an optimal solution: VM 2 is migrated to Host 3 before VM 1 is migrated to Host 2. As a side effect, this solution results in a high resource utilisation on both, Host 2 and Host 3, while Host 1 provisions no VM and thus can be set offline. The evolution of Algorithm B from the initial state to this final state can be observed in fig. 6.

5 Related Work

A number of approaches exist that aim at automating the management of virtual environments. In [13], the authors present *VIOLIN*, a policy-based system, which uses the Xen hypervisor for virtualisation. The system consists of one monitor daemon per physical machine and a central adaptation manager. The adaptation manager uses the data gathered by the monitor daemons to request changes in VM resource allocation.

In [6] Grit et al. present a second policy-based management approach. The authors use *Shirako*, a Java-based toolkit for dynamic resource sharing, to explore algorithmic challenges with regard to policy usage for adaptive VM hosting. Zhang et al. [20] describe a control-theoretical model for VM adaptation, based on the idea that each VM is responsible for adjusting its demand for resources, with respect to efficiency and fairness.

In [11], the authors use a feedback-control strategy to address dynamic resource allocation problems. They employ an infrastructure based on Xen, RUBiS and TPC-W. Here, time series analysis is used to forecast the behaviour of a virtualisation-based system. In [1] Bobroff et al. present a mechanism for dynamic migration of VMs based on a load forecast. Menascè et al. use utility functions for dynamic CPU allocation to virtual machines [10]. The authors test their approach by means of simulations that are based on historical data.

The previously presented approaches represent first steps taken by the research community to develop strategies for highly specialised, intelligent controllers. It is however hard to objectively evaluate and compare the presented strategies as the authors use different architectures, perform incommensurable tests and some even rely on simulations with historical data to test the performance of their strategies.

Currently, no commercial solutions exist that are capable of managing VM-based environments of QoS-based services in a fully autonomic way. Mainly, existing commercial solutions aim at supporting the work of system administrators, e.g. by providing user-friendly management interface (e.g. XenCenter, VMware VirtualCenter). However, *Novell Zenworks Orchestrator*[1] and *Platform Computing VM Orchestrator*[2] are two commercial products that support automated, rule-based migration of VMs in data-centre environments. In contrast to

[1] `http://www.novell.com/products/zenworks/orchestrator/data_center.html`
[2] `http://www.platform.com/Products/platform-vm-orchestrator`

In: Proceedings of the 15th Annual Workshop of HP Software University Association, Hosted by Al Akhawayn University in Ifrane, June 22 - 25, 2008

our approach, these products do not support alternative control algorithms and are currently not capable to optimise VM environments according to application-level QoS. A major advantage of using a modular architecture like ours is that control approaches like the ones in [13, 6, 11, 1, 10] can be easily adapted and integrated into the framework. This way, certain aspects of the approaches presented above can be reused.

6 Summary

We designed a modular framework for autonomic QoS management of VM-based application services, which separates the management APIs provided by individual VM technologies from the high-level controllers used for autonomic management. The framework allows an easy replacement of control algorithms and can be used as a common platform for automated VM management in data centres.

Future work will concentrate on several fields: autonomic control algorithms, enhancements to the framework and additional testing. The control algorithms presented in this paper are first examples that are only capable to deal with a single resource type. Thus, currently the VM Logic does not need to perform sophisticated bottleneck detection. Further research in pool-level control algorithms is necessary to tackle the presented multi-dimensional multiple knapsack problem. We are confident that the definition of constraints will allow us to simplify this problem and will lead to control approaches that at least provide a good solution. We plan to add support for other resources (e.g. network load) to the framework. This also requires an enhancement of the VM Logic component. In addition we plan to introduce a redundant Pool Manager component to eliminate this single point of failure. We plan to deploy the framework in the production environment of the department of computing for scalability testing. In addition we will work on the integration of the presented approach with an existing architecture for decentralised SLM of SOA workflows and services, which is currently being developed in our lab.

References

1. Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, 2007.
2. T. C. Bressoud and F. B. Schneider. Hypervisor-based fault tolerance. In *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, volume 29, December 1995.
3. Alan L. Cox, Kartik Mohanram, and Scott Rixner. Dependable unaffordable. In *ASID '06: Proceedings of the 1st workshop on Architectural and system support for improving software dependability*, 2006.
4. Y. Diao, F. Eskesen, S. Froehlich, J. L. Hellerstein, L. F. Spainhower, and M. Surendra. Generic Online Optimization of Multiple configuration Parameters With Application to a Database Server. In *Proceedings of the fourteenth IFIP/IEEE*

In: Proceedings of the 15th Annual Workshop of HP Software University Association, Hosted by Al Akhawayn University in Ifrane, June 22 - 25, 2008

Workshop on Distributed systems: Operations and Management (DSOM 2003), Oct 2003.

5. Ariel Felner, Sarit Kraus, and Richard E. Korf. KBFS: K-Best-First Search. *Annals of Mathematics and Artificial Intelligence*, 39(1), 2003.
6. Laura Grit, David Irwin, Aydan Yumerefendi, and Jeff Chase. Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*. IEEE, 2006.
7. Java Management Extensions. `http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/` (accessed 05/2008).
8. J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36, 2003.
9. Dan Marinescu. Design and Evaluation of Self-Management Approaches for Virtual Machine-Based Environments. Master's thesis, Wiesbaden University of Applied Sciences, Dept. Design, Computer Science, Media, February 2008.
10. Daniel A. Menasce and Mohamed N. Bennani. Autonomic Virtualized Environments. In *ICAS '06: Proceedings of the International Conference on Autonomic and Autonomous Systems*, 2006.
11. Pradeep Padala, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive Control of Virtualized Resources in Utility Computing Environments. In *EuroSys '07: Proceedings of the 2007 conference on EuroSys*, 2007.
12. M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *Computer*, 38(5), 2005.
13. P. Ruth, Junghwan Rhee, Dongyan Xu, R. Kennell, and S. Goasguen. Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure. In *IEEE International Conference on Autonomic Computing*, 2006.
14. TPC-W Java Implementation. `http://www.ece.wisc.edu/~pharm/tpcw.shtml` (accessed 05/2008).
15. Virtual Box. `http://www.virtualbox.org/` (accessed 05/2008).
16. VMware ESX. `http://www.vmware.com/products/vi/esx/` (accessed 05/2008).
17. S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart. An architectural approach to autonomic computing. In *Autonomic Computing, 2004. Proceedings. International Conference on*, 2004.
18. The Xen API. `http://wiki.xensource.com/xenwiki/XenApi` (accessed 05/2008).
19. Xen Source. `http://www.xensource.com/` (accessed 05/2008).
20. Yuting Zhang, Azer Bestavros, Mina Guirguis, Ibrahim Matta, and Richard West. Friendly Virtual Machines: Leveraging a Feedback-Control Model for Application Adaptation. In *VEE '05: Proc. of the 1st ACM/USENIX int. conf. on Virtual execution environments*, New York, NY, USA, 2005.