Qualitative Characterization of Quality of Service Interference between Virtual Machines

Ágnes Salánki, Imre Kocsis, András Pataricza, Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary
Zsolt Kocsis, IBM Center of Advanced Studies Budapest, Budapest, Hungary

Abstract

Virtualization based server consolidation is the fundamental technology in data centers supporting e.g. server cost reduction, cheap HA failover setups or green computing. Virtual machines running on a shared server should be in an idealistic case independent of each other both from the point of view of logical and performability aspects; however, isolation is typically imperfect. The paper proposes a methodology for estimating parasitic interferences between virtual machines running on the same shared host by analyzing the core factors by means of data mining. The consequences to deployment policy are shortly addressed.

1 Introduction

Platform-level virtualization in the data center enables resource usage consolidation. The main means is the largely transparent migration of physical servers to *virtual machines* (VMs), hosted by a (typically) lesser number of *host machines*. The principle of resource reduction is the merging of all computational resources into a single pool. This results on the capacity assurance side in a substitution of the total of worst case capacity needs of the tasks statically deployed to the individual servers with that of the maximum of total of concurrently running applications (minus the virtualization related overhead).
Maybe equally importantly, modern virtualization technologies became a cornerstone of implementing adaptivity and autonomicity in the data center by supporting such management actions in the platform as the live migration of machines between hosts, suspending/resuming virtual machines, providing machine state level checkpointing and the fast instantiation of "clone" virtual machines from checkpoint images.
Through these actions, virtualization platforms provide excellent support on the virtual machine level for

a) modular redundancy based structural fault tolerance patterns,

b) many important error recovery patterns (as e.g. rollback or failover)

c) and c) some patterns mitigating errors caused by overload faults.

(For a taxonomy of the patterns of fault tolerant software, see [8].) Also, with a usually modest amount of dynamically reallocable redundant resources (a "resource pool" of hosts) in the system, the dependability measures can be readjusted on-demand.
However, virtualization introduces an additional resource arbitration layer in the form of the *Virtual Machine Monitor* (VMM) or *hypervisor*. Platform virtualization technologies can be divided into two categories: solutions with *bare-metal* and *hosted* architectures. In the hosted case, the virtualization technology runs on a regular operating system (some widely used examples are VMware Workstation, VirtualBox and Microsoft Virtual PC), while the VMM of bare-metal architectures is not insulated from the hardware by another software layer (VMware ESX, Xen).

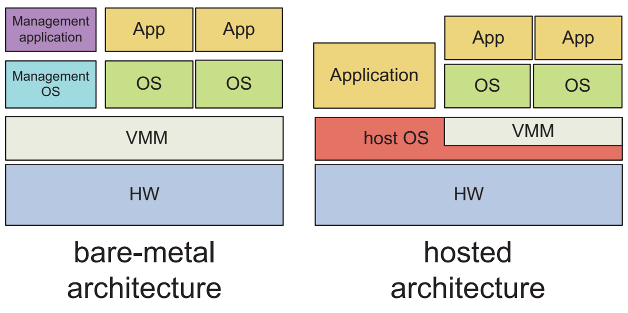

Figure 1 Bare-metal vs hosted architectures

The hypervisor presents virtual machines with a virtual operating platform. Fault isolation between the VMs is a crucial aspect of ensuring dependability in virtualized environments; this includes ensuring the lack of unacceptable *performance interferences* in the context of resource access arbitration.

2 Performance interference of VMs

Software stacks that in the native case would run on dedicated hardware have to share the physical resources of the host system in a virtualized environment, the access being governed by the scheduling policies of the VMM. Unsurprisingly, the performance of the services is affected by the *interference* caused by other virtual machines running on the same host. A virtual machine shows different service-level performance characteristics – more precisely, a different relationship between load and *Quality of Service* (QoS) – in most cases when it runs "on its own", than when multiple virtual machines share the same host. [1] and [2] provide empirical data on this phenomenon.

[1] also establishes a taxonomy for the sources of the interference: a) shared resourc*es visible* to the VMM directly or through performance counters – for instance, core allocation or memory capacity; b) shared resources that are essentially *invisible* to the VMM by their nature – e.g. cache space and memory bandwidth and c) the specifics of the virtualization technology and the scheduling disciplines employed.

Modern platform virtualization technologies as VMware ESX/vSphere and Xen provide support for enforcing explicit, allowance-based hard limits (or "caps") on the visible resource metrics. These are *CPU usage*, *physical memory usage*, *I/O Operations per Second* on a per-VM basis and *average bandwidth*, *peak bandwidth* and *burst size* for (virtual) network port groups in the case of ESX/vSphere 4. These complement priority-based scheduling options for the critical core subset of the resources.

Hard caps are commonly used in the industrial practice to establish performance isolation between virtual machines running services with stringent QoS requirements. Caps are also intended to serve as a form of resource usage fault containment in these cases.

However, a number of problems arise when hard capping is used as a mechanism for performance isolation in QoS-controlled environments. A significant technical factor is that these mechanisms confine only the use of visible resources and they lack providing a guaranteed isolation from interferences through invisible resources.

Moreover, even the platform level supervisory and resource-to-job allocation services have a complex interdependence with the application level resource usage pattern. The QoS of the "resource services" depends on these interferences– e.g. the patterns of parallel access influence the execution time in varying ways through caches, or operation delay for storage I/O. The service-level QoS of the virtual machines in turn depends on these parameters.

Establishing a performance model for services that leads to proper capping parameters is an ill-supported activity, even without taking interferences into account. Naturally, performance modeling is the starting point for performability design of HA applications. Although there is a large body of approaches for resource use forecasting in computing systems (for a best practice guide, see e.g. [3]), these models and model identification methods tend to be

a) quite complex;

b) establish fine-granular models, thus necessitating additional effort for linking the model to system management policies and event handling (see later), and

c) focus on a fixed operational point or heavily constrained operational domain, while in virtualized infrastructures the on-demand *migration* of virtual machines between hosts and *restructuring* of capping on the individual machines can change these significantly.

2.1 Qualitative characterization

System management in general deals with QoS-management in highly qualitative terms; only a few service QoS and service load classes are defined for capacity planning purposes and the supporting coarse-grained, "good enough" resource allocations are sought. VM performance interferences in this setting manifest as effects that modify the provided QoS class of a VM running on a host in itself to another one when other VMs are also run on the host.

If known, these constraints can then be taken into account during configuration planning – e.g. at the most basic level by prohibiting deploying two VMs to the same host when QoS class changing interferences can occur. Note that for a heterogeneous set of VM types – as is usual in enterprise data centers – not all VM types will show radical performance interferences. We can reasonably expect a CPU-bound application server and an I/O bound file server not to interfere with each other to a great extent (under normal load). However, explicitly testing all possible VM-to-host deployments for service QoS category level interferences is clearly not a tractable approach.

2.2 Estimation from observations

Virtualization platforms provide a set of built-in counters that collect various virtual machine and platform level run-time data – and to that, partially at a more fine-grained level than at which caps are specified. For instance, ESX/vSphere 4 has counters for disk commands issued, aborts, disk read and write requests while the I/O per second limit does not distinguish operations.

We propose that the platform counters enable a conservative, phenomenological way of modeling possible interferences by formulating relationships between QoS-effects and effects on the platform counters.

These counters enable estimating cross-VM interference.

1. At first, a measurement of the QoS and platform characteristics of a single VM running on a host alone delivers its performance characteristics.
2. Repeating the measurement with (at least) another VM also present as "disturbance", we arrive at a set of QoS differences accompanied with a correspond-

ing set of platform metric differences – these will serve as *empirical proofs of sensitivity*.

3. Using classic data mining techniques, we can establish the platform metric difference categories that can be best linked to categorical changes in the QoS.
4. These factors of high sensitivity in turn can be used for a conservative solution exclusion strategy during configuration planning.

For instance, if the experiments indicate that a "high" change in the disk write rate compared to the case when the measured VM ran alone invariably leads to a significant QoS-change, than this finding provides a supporting argument for not deploying beside this VM another that is likely to generate at least so big a disk write rate.

3 Experimental environment

We have developed an experimental environment to study our proposed approach. The environment was inspired by the two currently most prevalent virtualization benchmarks, VMmark [4] and vConsolidate [5].

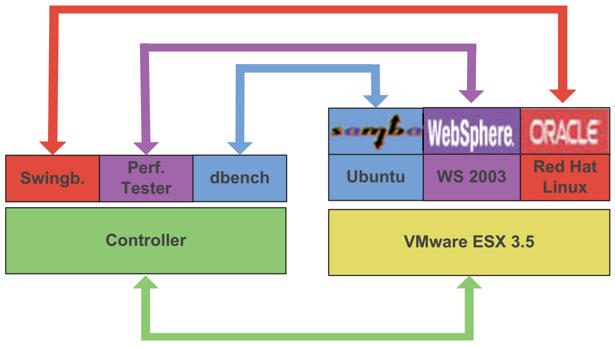

Figure 2. Experimental environment

Three types of virtual machines are examined. 1) A VM running an Oracle database server, loaded with the de-facto industrial benchmark Swingbench; 2) A VM running a Websphere Application Server and its "Plants by Websphere" example application, loaded with a custom page visit sequence that is replayed by IBM Rational Performance Tester [6]; and 3) a Samba server VM, loaded by the file server benchmarking tool dbench. The QoS metrics of the services are average database query response time, average web page response time and average server-side data throughput, respectively. The system running he measured virtual machines is a Fujitsu Siemens Primergy TX 120 machine, with a dual core, 2,66 GHz Intel Xeon processor and 2GB RAM. As a hypervisor, the host runs VMware ESX 3.5.
The load generator and service QoS registering components were run from the same tester host (a machine with significantly more resources than the one running the hypervisor) and we use nonstandard applications and loads. This way, our results are to serve only the initial evaluation of experimental approaches and are unambiguously unusable for performance comparison purposes. Consequently, we give here only normalized values in the results.
Experiments are run by a custom Java test harness on the tester machine that automates environment setup/restore, load generator management, the collection of results (QoS as well as platform metrics) and the batch execution of experiments. (See **Figure 2**.)

4 Baseline and interference experiments

A virtual machine/load generator pair has three configuration parameters in each experiment: the CPU limit of the virtual machine (hard upper cap), the memory limit of the virtual machine and the number of simulated users for the respective load generator. For each virtual machine type, the limits range from 10% to 90% in steps of 20%; three user number values are defined in each case, with 10 being the lowest and 50 being the highest uniformly.

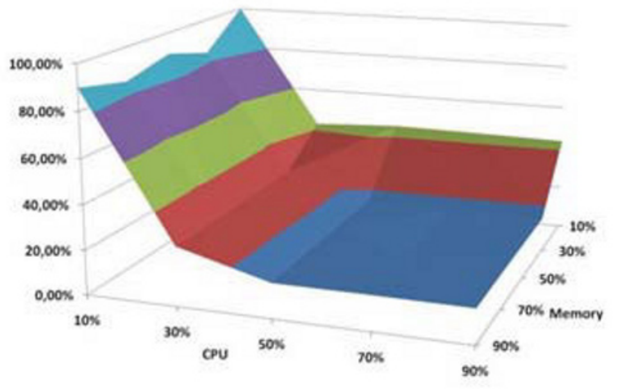

Figure 3. Database running alone; normalized performance w.r.t. limits (10 users)

We have conducted two series of experiments. In the first, each VM was measured as the sole running one on the host with each possible configuration setting. In the second, all three *pairs* were measured, with all possible configuration settings (limits were selected so that for the CPU as well as memory, the corresponding limits of the two VMs added up to 100%). Accounting for the cases in the second series where the services in the VMs consistently did not even start or crashed, 765 experiments were run successfully. Each experiment ran for 20 minutes.
The first series of experiments gives a "baseline" steady-state performance estimation for each VM, mapping the configuration space to QoS and platform metric measurements. As an example, **Figure 3, 5** and **6** show the QoS-surface of the database server with respect to CPU

and memory limits (10, 20 and 50 users, respectively). As expected, under the given load the server can become CPU as well as memory-constrained through too stringent limits; however, note how overlimiting the CPU has far more serious consequences when the system is not saturated in the 10 user case.
We chose to use limits only on the CPU and memory on the one hand to expedite the measurement campaign and on the other hand to first examine such cases where interference through visible resources is possible. Further research will examine cases, where all limitable visible resources are capped.

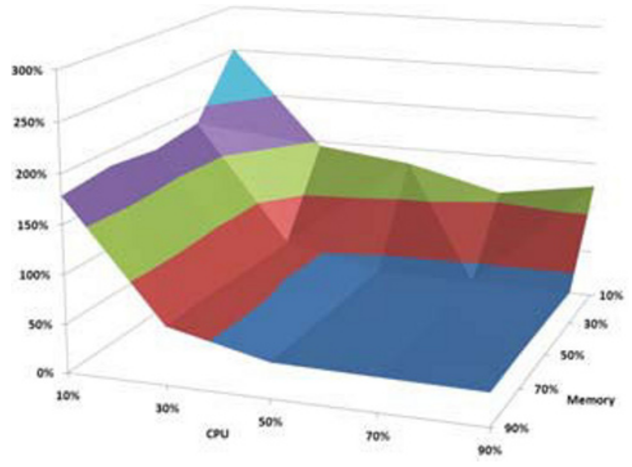

Figure 5. Database running alone; normalized performance w.r.t. limits (20 users)

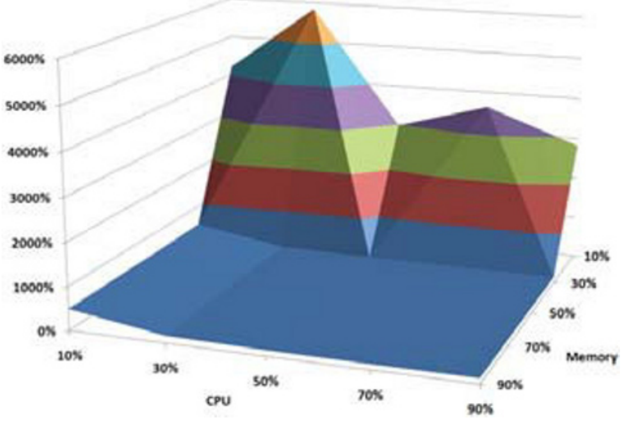

Figure 6. Database running alone; normalized performance w.r.t. limits (50 users)

From the second set of experiments it is directly discernible that depending on the operational regions of the VMs (as defined by capping and service load), there are significant performance interferences; see for example **Figure 4**. It can be seen that for a heavily CPU-constrained database (where the QoS is already degraded to some extent), the interference effect is very significant. This indicates that in virtualized environments the impact of allocating an insufficient amount of slack to performance-critical services can have a magnified impact w.r.t. native ones.

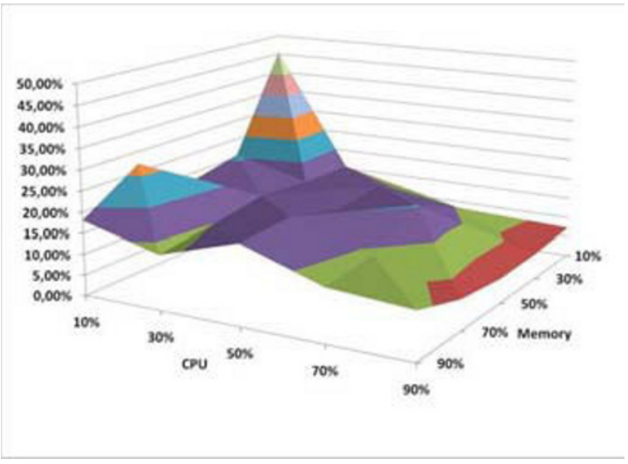

Figure 4. QoS-change of database VM (10 users) w.r.t. Figure 3 – disturbance: dbench (10 users)

5 Data mining interference measurements

The constructive use of virtualization has to take into account parasitic coupling between different applications through the resources. Note that this is fundamental from the point of view of modular replication based dependability assurance of applications – a technique that can be exploited to a great extent in virtualized environments, due to the ease with which virtual machine replicas can be created. Mainstream technologies for assuring dependability (typically high availability) exploit these capabilities.
However, the cross-coupling observed during the measurements between functionally independent applications may corrupt these schemes; moreover, it may introduce a nondeterministic, thus nonpredictable cross-coupling invalidating QoS guarantees.
Artificial intelligence offers effective methods for the estimation of principal factors and their impact by sophisticated algorithms, embedded in data minig tools. We have performed numerous data mining experiments on QoS and platform metric differences.
We have found that building *decision trees* – more specifically, *classification trees* with QoS change classes in the leafs – are a promising way to capture the most important factors in the “footprint” of interferences on the platform level.
These decision trees have the advantage of clearly separating the qualitatively different domains of operations together with the principal factors dominating their main

characteristics. We use a built-in classifier (J48) of the popular machine-learning suite WEKA [9].
Let us consider as an example the database server with the 10 user load class (**Figure 3**). We take as disturbance the co-allocation of the virtualized application server to the same physical host in all CPU/memory limit points with all three load classes. The decision tree on the platform metric differences w.r.t. the case when the database VM runs alone on the host is depicted on **Figure 7**.
The nodes of the tree are the differences in *platform* metrics (that is, aggregate ones for the platform and not VM-specific). For now, the classification of the QoS-difference into classes is performed automatically, in a preprocessing step. The ratio of properly classified elements and improperly classified elements – if any – is displayed in parentheses in the leafs of the tree on the figure. Although we do not wish to represent quantitative results, for the reference the most serious QoS difference ratio class ("`High`") lies in the x2 – x3 range, while the class `Small` represents cases where the multiplier is smaller than 1.4.

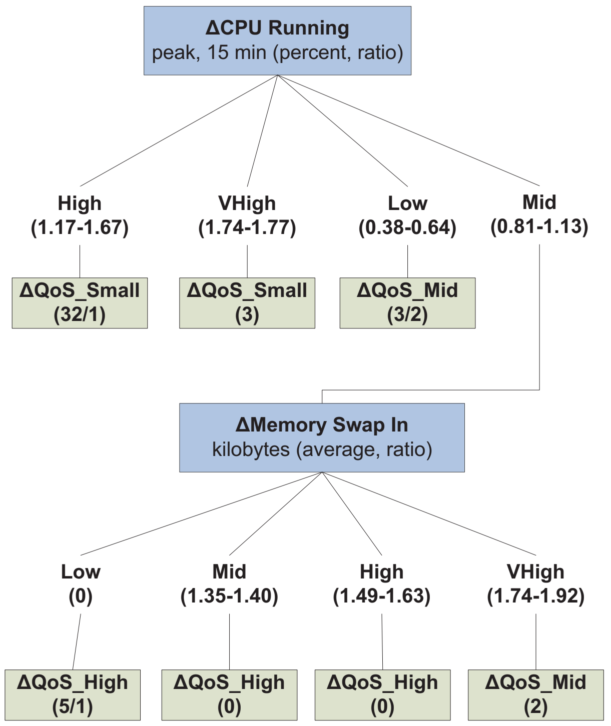

Figure 7. Classifying the QoS-change for the Oracle VM; disturbance: WebSphere

In VMware ESX, the `CPU Running` (15 min peak) metric represents the percentage of time a CPU group ran at peak over a period of 15 minutes. The counter `Memory Swap In` measures the total amount of memory swapped in.
The interpretation of the results is the following.

1. In most operational points, the database server is largely unaffected by the (usually heavily CPU-bound) application server.
2. However, there were cases when the amount of peak usage of the CPU does not really change – still, significant interferences were observed.

As another example, let us consider the case when the disturbance is the dbench virtual machine (**Figure 8**; note that here using differences are used instead of ratios, as due to the distribution of the data it supports decision tree building better).
Here the most serious QoS difference class ("`+High`") lies in the tens of seconds range, while the class "`Small`" represents deviations smaller than a half second. Interestingly, two cases were found where the QoS was actually slightly improved in the two-VM case.

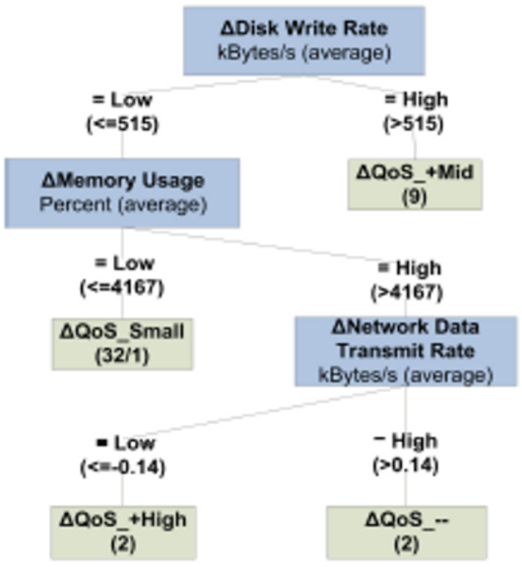

Figure 8. Classifying the QoS-change for the Oracle VM; disturbance: dbench

In VMware ESX, `Memory Usage` is defined as the ratio of the total physical memory consumed (by VMs and the virtualization environment) and the amount of physical memory installed.
This way, the decision tree has the following interpretation:

1. Even relatively small changes in the host-level disk write rate lead to measurable changes in the QoS (recall that I/O throughput was not capped).
2. When the disk write rate is not significantly influenced, the QoS virtually does not change, provided

that the change in actual host memory usage remains below approximately +40%.

A consequence to be drawn from the example is that it is ill advised to use the remaining layer of the decision tree on an as is basis, as for both categories there are only two underlying measurements at the "edge" of the configuration and operational space mapped out by the measurement campaign. At the same time, the two examples clearly indicate the limitations of automated data processing methods. Even simple visual comparison indicates heavy differences at the phenomenological level; however, the estimation of the causal roots needs a somewhat speculative expert interpretation. Here we try to trace back the observed phenomena in the particular subdomain to the resource usage level as most probably such interferences originate in (nearly) saturated resources. This needs a deep understanding and interpretation of the dynamics of the resource use.
As such, for prediction purposes we can employ the technique of *pessimistic overabstraction* of the behavior of the system and state that significant changes in the overall memory usage (more than approximately +40%) *may* lead to significant QoS changes.
These pessimistic overabstractions form the basis of a cautious, pessimistic design paradigm. More measurements and refined decision trees may reduce the unwanted resource underexploitation originating in the pessimism of the approach. Most importantly, future work will add the CPU/memory limit and load characteristics to the platform metric differences as the supporting variable set of decision tree building, in order to incorporate in the decision trees one aspect our initial methodology does not deal with - the fact that the presence and causes of interference phenomena may depend on load and the amount of available resources.

6 Interference and configuration planning

The most basic aspects of configuration planning in virtualized environments is designing *deployment* – which virtual machine should be instantiated on which host – and deciding on the resource allowances – the parameterization of various capping and scheduling options – on a per host basis. Similarly to classic configuration planning approaches, the objective function of configuration planning in virtualized environments can be based on various service-level requirements as e.g. the need to adhere to SLAs, the objectives of risk management and operational cost constraints. Even some novel dependability issues such as the fault of a single physical machine becoming a common-mode fault for multiple services do not pose fundamentally new questions.
Multiple works have examined configuration planning of virtualized environments with the objective (or one of the objectives) being maintaining constraints on the QoS of the services provided. Most of these works approach the problem as a purely combinatorial (vector packing) or mixed integer linear programming one (see e.g. [10][11][12]). One of the common characteristics of these known approaches is that they do not deal with performance interferences of any type.
Taking performance interferences into account during configuration search necessitates the ability to approximate the QoS of a virtual machine instance given its CPU/memory limits and its load. Even without virtualization and parasitic interferences, this topic can become very involved when the granularity of the underlying time window of the QoS metric is fine, or the precision of the required estimation is big, especially when comparatively long prediction horizons are taken. The complexity of QoS prediction further grows with the introduction of varying and/or highly nondeterministic loads. (Again, see e.g. [3].) In the context of our current work QoS is defined as a steady-state metric with a large underlying time window (tens of minutes). Admittedly, for the most important "averaging" QoS metrics this will mask the (potential) presence of outliers and transients phases with significant amplitude. However, the usual industrial practice largely follows the same approach at *design time* as building more precise performance models prior to the operational phase is usually infeasible – additionally, the portability of performance assumptions between operational settings tends to be poor.
We have conducted a series of single-machine performance experiments that map out the QoS of individual virtual machines in a space of points of operation defined by load, CPU and memory limits. By applying qualitative categorization, these give rise to a set of "operational mode maps".

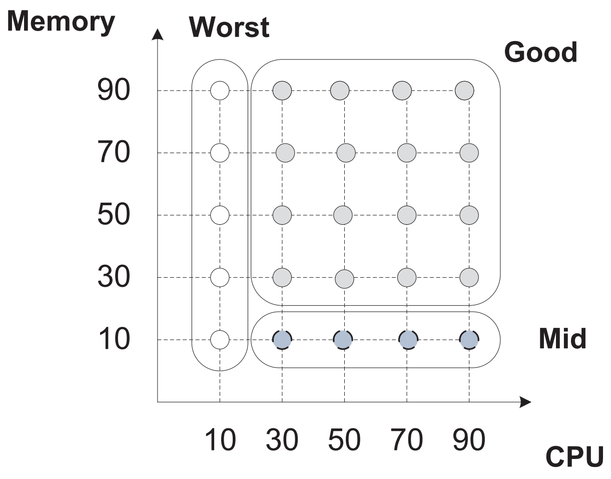

Figure 9. An example qualitative categorization of **Figure 3.**

Also, we have performed a series of interference measurements between pairs of virtual machines that provide empirical evidence for the deviations from the single-VM

maps – when only the two machines run on the same host and the CPU and memory are partitioned between them.
However, supporting configuration planning by direct measurement only is clearly a process that does not scale. Denoting the set of virtual machines with N and disallowing virtual machine replication on the same host, worst case there are 2^N virtual machine combinations that would have to be measured in all possible resource partitioning and load combinations.
Instead, we propose qualitatively estimating the QoS of the service provided by a VM in multiple-VM configurations by approximating the qualitative QoS *difference* the other VMs may cause. Let vm_a be the virtual machine for which we seek its expected QoS class $Q_e$under load l and with CPU and memory limits c and m. The outline of the approach:

1. An "undisturbed" QoS class Q_u is computed from the single-VM measurements of vm_a via linear approximation for the operational point (l, c, m).
2. The platform metric difference category vectors that can lead to significant changes (more precisely, change categories) in the QoS of vm_a are gathered from decision trees that have been inferred from the interference experiments for vm_a.
3. For the set of other VMs to be deployed on the same host, it is computed that what their aggregate impact on these platform metrics can be in their respective operational points, based on their single-VM measurements.
4. Q_u is modified with the QoS-category of the interference relation(s) that is compatible with this impact.

Note that the third step necessitates a mapping between the VM-specific metrics (e.g. VM memory usage) and the platform metrics (host memory usage). However, establishing this mapping – using pessimistic approximation, if necessary – is quite straightforward for most platform metrics.
Estimating the impact of other machines this way is not guaranteed to deliver exact or even always correct results. Still, this procedure is appropriate to conservatively *discard* such configurations during planning where there is evidence that a setup with a given platform-level metric signature will lead to unacceptable QoS of one or more of the virtual machines hosted, despite their individual single-VM behavior. Naturally, accepted solutions have to be performance-tested to find those that were not discarded, but are inappropriate QoS-wise.

7 Outlook: monitoring configuration planning

Using data mining techniques on the results of single-VM measurement campaigns promises to lead to a method for the deeper understanding of the internal characteristics of virtual machines and eventually to synthesizing monitoring configurations that are defined in terms of hypervisor metrics, not guest OS ones and direct service monitoring.

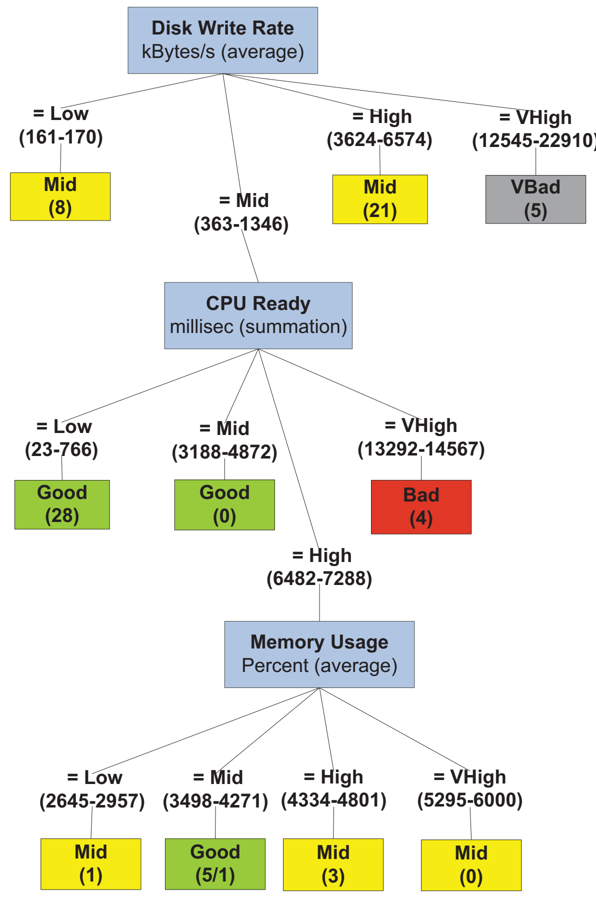

Figure 10. Decision tree built on the VM specific metrics of all database experiments

Figure 10 depicts the decision tree built on the experimental data gained from all single-VM database experiments. This tree contains the three phenomenologically most important factors defining the performance of the database server, *regardless of the load and capping employed*. (There is roughly one order of magnitude difference between the QoS categories in the leafs.) It is directly discernible that in the operational points we have data for, neither low nor high disk write rates do accompany optimum performance. Even when the disk write rate is what is expected under normal operations, a high or very high `CPU Ready` time – the time a virtual machine has to wait before it is scheduled for execution on a CPU – is associated with degraded performance in more than half of the cases. At the very least, such decision trees provide a solid initial monitoring configuration that can be refined via online learning and is not invalidated by online data center reconfigurations.

8 Conclusion and future work

The experimental validation and refinement of the interference-estimation methodology is currently ongoing work; here our initial results were reported. Notably, we have to address the open question that how much "disturbance" measurement campaigns are necessary for a moderately extended set of virtual machines. Also, as an implementation a mixed integer linear program is being designed for the interference-aware configuration planning of virtualized environments.

9 Literature

[1] O. Tickoo, I. Ravi, I. Ramesh, and D Newell, "Modeling virtual machine performance." ACM SIGMETRICS Performance Evaluation Review 37, no. 3 (2010): 55.

[2] J.N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. Mccabe, and J. Owens, "Quantifying the Performance Isolation Properties of Virtualization Systems," Proceedings of the 2007 workshop on Experimental computer science, 2007, pp. 13-14.

[3] Günther A. Hoffmann, Kishor S. Trivedi, and Miroslaw Malek. "A Best Practice Guide to Resource Forecasting for Computing Systems." IEEE Transactions on Reliability 56, no. 4 (December 2007): 615-628.

[4] V. Makhija, B. Herndon, P. Smith, et al. VMmark: A scalable benchmark for virtualized systems. VMware, Inc., Tech. Rep. 2006.

[5] K. Shi, J. P. Casazza, M. Greeneld, "Redefining server performance characterization for virtualization benchmarking," Intel Technology Journal, vol. 10, 2006, pp. 243-252.

[6] IBM Rational Performance Tester product page. http://www-01.ibm.com/software/awdtools/tester/performance/

[7] dbench home page. http://dbench.samba.org/

[8] R. Hanmer, Patterns for Fault Tolerant Software, John Wiley & Sons, 2007.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten, "The WEKA Data Mining Software: An Update", SIGKDD Explorations, Volume 11, Issue 1.2009.

[10] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application Performance Management in Virtualized Server Environments." Network Operations and Management Symposium 2006 NOMS 2006 10th IEEEIFIP, 373-381. IEEE.

[11] V. Petrucci, O. Loques, and D. Mossé, "A dynamic optimization model for power and performance management of virtualized clusters." Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking, 2010, pp. 225-233

[12] M. Cardosa, M.R. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments", IFIP/IEEE International Symposium on Integrated Network Management, 2009, pp. 327-334