Database Virtualization: A New Frontier for Database Tuning and Physical Design

Ahmed A. Soror   Ashraf Aboulnaga   Kenneth Salem

David R. Cheriton School of Computer Science
University of Waterloo
{*aakssoro, ashraf, kmsalem*}*@cs.uwaterloo.ca*

Abstract

Resource virtualization is currently being employed at all levels of the IT infrastructure to improve provisioning and manageability, with the goal of reducing total cost of ownership. This means that database systems will increasingly be run in virtualized environments, inside virtual machines. This has many benefits, but it also introduces new tuning and physical design problems that are of interest to the database research community. In this paper, we discuss how virtualization can benefit database systems, and we present the tuning problems it introduces, which relate to setting the new "tuning knobs" that control resource allocation to virtual machines in the virtualized environment. We present a formulation of the virtualization design problem, which focuses on setting resource allocation levels for different database workloads statically at deployment and configuration time. An important component of the solution to this problem is modeling the cost of a workload for a given resource allocation. We present an approach to this cost modeling that relies on using the query optimizer in a special virtualization-aware "what-if" mode. We also discuss the next steps in solving this problem, and present some long-term research directions.

1. Introduction

Technologies for *resource virtualization* are currently being employed at all levels of the IT infrastructure to improve provisioning and manageability, with the goal of reducing total cost of ownership. This means that database servers – like other types of software systems – will increasingly be run in virtualized environments, inside virtual machines. This raises two new questions that need to be addressed by the database research community: (1) How can the capabilities provided by virtualization environments be leveraged to improve the deployment, flexibility, and resilience of database systems? and (2) How should the administration, management, and tuning of database systems change to adapt to the new world of virtualized environments? Both of these questions are of interest to database researchers. In this paper, we focus on the second question, and we present preliminary results of our work towards addressing it.

1.1. Why Virtualization?

The basic idea behind resource virtualization is to decouple the user's perception of hardware and software resources from the actual implementation of these resources. Resource virtualization technologies add a flexible and programmable layer of software between "applications" (in our case database systems) and the resources used by these applications. This layer of software maps the virtual resources perceived by applications to real physical resources. By managing this mapping from virtual resources to physical resources and changing it as needed, the virtualization layer can be used to transparently allow multiple applications to share resources and to change the allocation of resources to applications as needed. This makes resource virtualization an excellent tool for flexibly deploying and managing database servers and other software systems (such as web servers or application servers) [1].

One of the most prominent examples of resource virtualization is *machine virtualization*, which is the focus of this paper. Machine virtualization technologies, such as Xen [3, 23] or the different VMware products [21], provide the capability of creating different *virtual machines* that share the resources of one real physical machine (or sometimes even many machines [20]). Each one of these virtual machines runs its own separate operating system, and the operating system and applications in each virtual machine perceive that they are using dedicated machine resources (CPU, memory, and I/O), whereas in reality the physical resources are shared among all the virtual machines. The virtualization layer, known as the *virtual machine monitor* [15, 17], controls the allocation of physical resources to virtual machines and can change it as needed. The virtualization layer also provides other capabilities such as the ability to save an *image* of a running virtual machine, and to restart the virtual machine from a saved image on the same physical machine or a different one. Some virtualization layers even provide the ability to migrate live, running virtual machines between physical machines [5, 16].

The capabilities provided by machine virtualization technologies can potentially be used to reduce the total cost of ownership of software systems, including database systems. For example, the ability to save virtual machine images can be used to simplify the process of software distribution and deployment. A virtual machine image can be created with a software system and any other required software modules already installed and running. A user can then deploy the full software system by simply copying the virtual machine image and starting the saved virtual machine. Such ready-to-deploy virtual machine images are known as "software appliances," and there are currently repositories of these appliances that can be found on the web [19]. This software appliance model eliminates many of the problems of software installation and makes it possible to flexibly deploy the same software system on different machines. It is easy to see that it would be very useful to have a "database appliance" that is ready to deploy on different physical machines.

Another benefit of machine virtualization is in the area of *server consolidation*. An organization typically has different software systems for different operational and business activities. Instead of having different server machines for the different software systems, each provisioned for peak load, we could run the software systems in virtual machines and have the virtual machines share the same physical resources, thereby reducing the amount of hardware that needs to be provisioned, operated, and maintained. Since organizations typically have multiple database servers, it is easy to see that database systems would stand to benefit from such server consolidation.

1.2. Tuning and Virtualization

In addition to the two examples above, there can be other ways for database systems to benefit from virtualization. Indeed, developing ways for database systems to leverage the capabilities of virtualization is in itself an interesting research problem. However, in this paper we focus on another class of interesting research problems: How to tune a database system and the virtual machine in which it is running, starting from deployment and throughout operation.

Virtualization introduces some new "tuning knobs" that affect the performance of the database system. These tuning parameters must be set either by the database administrator or automatically in a self-managing way. The most important of these tuning parameters are the ones that control the share of each physical resource (CPU, memory, and I/O bandwidth) that each virtual machine gets. These virtual machine parameters also interact with database system tuning parameters such as buffer pool sizes, which means that the two sets of parameters should be tuned simultaneously.

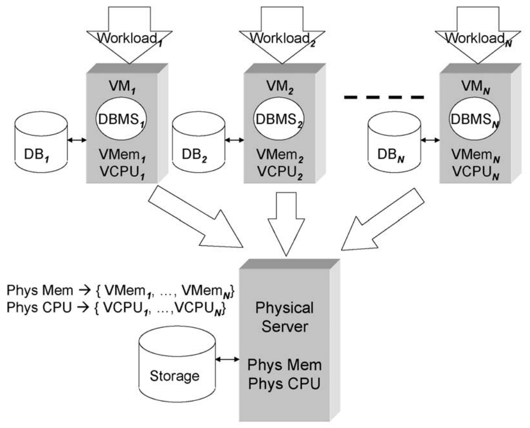

Figure 1. Tuning and virtualization.

The importance of tuning these virtual machine parameters becomes apparent as soon as we decide to run multiple database systems inside virtual machines that share physical resources. We are immediately faced with the question of how to allocate physical resources to each virtual machine. This *static* version of the problem, which we refer to as the *virtualization design* problem, is our focus in this paper. The virtualization design problem can be defined as follows: *"Given N database workloads that will run on N database systems inside virtual machines, how should we allocate the available resources to the N virtual machines to get the best overall performance?"* In this context, we use *throughput* as our measure of performance (i.e., we are interested in using the least amount of resources necessary to fully process a given query).

We face the virtualization design problem, for example, when we are consolidating multiple database servers onto one physical machine by running each database server in a virtual machine (Figure 1). In this paper, we motivate the virtualization design problem and present a general framework for its solution. An important challenge in solving this problem is how to model the performance of a database system inside a virtual machine. We present an approach for addressing this challenge that relies on using the query optimizer in a new "what-if" mode that is virtualization-aware. We also discuss the next steps in solving this problem and other, more general virtualization tuning problems.

Note that we face the virtualization design problem whenever we run multiple virtual machines on the same physical machine, regardless of the applications running in the virtual machines. However, we may be able to find a better solution to the problem when these applications are database systems due to several factors. First, database workloads consist of SQL queries with constrained and highly stylized resource usage patterns. Second, these queries are highly variable in the way they use resources. For example, some queries heavily use CPU while others heavily use I/O. This makes database systems sensitive to – and able to benefit from – changes in resource allocation. Third, database systems already have a way of modeling their own performance, namely the *query optimizer*.

Note also that the virtualization design problem is similar to the problem of capacity planning that is already faced by database users even without considering virtualization: How much resources do I need (or what kind of machine should I buy) for a given database workload? However, our problem is much more constrained because the underlying machine parameters are fixed in advance. This makes the problem more tractable than the general capacity planning problem, which is notoriously difficult to solve.

The rest of the paper is organized as follows. In Section 2, we present an overview of related work. Section 3 outlines the virtualization design problem. Sections 4 and 5 describe our approach to cost modeling. Section 6 presents an experiment illustrating the effectiveness of our cost modeling. In Section 7 we present discussions and next steps. Section 8 concludes the paper.

2. Related Work

We are not aware of any work on the problem of virtualization design, whether for database systems or other types of applications. However, our work builds on the large body of recent work on providing the necessary infrastructure for machine virtualization [3, 15, 17, 21]. Our work is also related to work on capacity planning and allocating resources to applications in non-virtualized environments [9], although this work cannot easily be extended to the problem of determining an optimal sharing of resources, especially for database workloads in virtualized environments. Tuning database systems for the available resources has been the focus of much work [22], but in this paper, we are tuning the *resources* to the *database system*. And since we change the resources available to the database system, work on automatically adapting the database system to the available resources is relevant [2, 7, 10, 13, 18].

3. Virtualization Design

In this paper, we focus on the problem of statically allocating physical machine resources to N different virtual machines running N different database workloads. We refer to this problem as the *virtualization design* problem, and we view it as an extension of database physical design. In non-virtualized environments, determining the best physical design for a database given a workload involves setting parameters such as how much memory can be used for the database and how this memory is partitioned into different memory pools. In virtualized environments, we can also set the amount of resources given to a database workload (more precisely, we can set the amount of resources given to the virtual machine that contains the database system running the workload). These resource settings are physical parameter that affect performance, and hence we consider setting them to be part of the physical design process.

Setting the resource allocation parameters becomes challenging only if there are multiple virtual machines running different database workloads and sharing the same physical machine. If there is only one virtual machine, then that machine should get all the available physical resources. If there are multiple virtual machines but they are all running similar database workloads, then the available resources should be divided equally among the virtual machines. The challenge is if the different virtual machines are running workloads with different characteristics and different resource usage profiles. It is common for different database workloads to have different resource usage profiles, which makes the virtualization design problem relevant and challenging for database systems.

The virtualization design problem can be formulated as follows. We are given N database workloads, $W_1, \ldots, W_N$, where each workload is a sequence of SQL statements against a separate database. The N workloads will be run on database systems that reside in N different virtual machines. For simplicity, we will assume that we are using different instances of the same database system for the different workloads. The virtualization environment allows us to control, for m different physical resources, the share of each resource that is allocated to each virtual machine. Examples of resources that can be controlled in this way are CPU, memory, and I/O bandwidth. Each workload gets a *fraction* of each resource. We denote the fraction of resource j that is allocated to workload W_i by r_{ij}. We denote the overall resource allocation for workload W_i by a vector $R_i^T = [r_{i1}, \ldots, r_{im}]$. The fractions of all resources allocated to all workloads can be represented as an $m \times N$ matrix, which we denote by $\mathbf{R}$. For each workload, W_i, there is a cost for running the workload that depends on the resource allocation, R_i. We denote this cost by $Cost(W_i, R_i)$, and our goal is to minimize the overall cost for all the workloads. Thus, our problem is to find

$$\arg\min_{\mathbf{R}} \left(\sum_{i=1}^{N} Cost(W_i, R_i) \right)$$

subject to $r_{ij} \geq 0$ for all i, j, and $\sum_{i=1}^{N} r_{ij} = 1$ for all j.

This is a combinatorial optimization problem, and the framework to solve it is presented in Figure 2. To solve this problem, we need (1) a search algorithm for enumerating candidate solutions, and (2) a method for evaluating the cost of a candidate solution. For the search algorithm, we should be able to use any standard combinatorial search algorithm

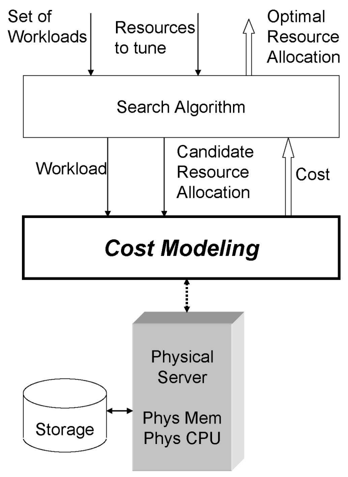

Figure 2. Framework for virtualization design.

such as greedy search or dynamic programming. What is more interesting, and unique to virtualized environments, is the modeling of the cost of a database workload for a given resource allocation, i.e., the function $Cost(W_i, R_i)$. We focus on this function for the remainder of the paper.

4. Cost Modeling

Modeling the cost of a workload for a given resource allocation is important for all kinds of workloads, not just for database workloads. However, database systems have the unique advantage that their *query optimizer* already provides a model for the cost of workload execution. Thus, our approach is to make the query optimizer aware of virtual machine resource allocation, and to use the optimizer to estimate the cost of workload execution for different levels of resource allocation in the virtual machines. This follows the same lines as traditional physical design work, which also uses the query optimizer for cost modeling. The query optimizer is especially suitable for modeling costs in the virtualization design problem, since query optimizers typically optimize for *minimizing total resource consumption*, which corresponds to our goal of maximizing throughput.

Thus, our value for $Cost(W_i, R_i)$ is the sum of the estimated execution times for all the queries in W_i as computed by the query optimizer in a special mode in which it is aware of the resource allocation, R_i, and its effects on query performance. What remains now is defining this new "what-if" mode in which the optimizer can model the effects of a particular level of resource allocation to the virtual machine. We denote the level of resource allocation by R.

To define this new mode, we note that the query optimizer models costs and chooses query execution plans based on (1) the available access paths, (2) database statistics, and (3) parameters that describe the physical environment. The first two factors are not affected by changes in R, but the third factor is significantly affected by such changes. Thus, to make the optimizer aware of the effect of a certain R, all we need to do is find the appropriate values for the parameters that the optimizer uses to model the physical environment. We refer to this set of optimizer parameters as P. Once we set P for a given R, we can simply optimize the workload queries with this P, using the available access paths and database statistics without change, and we can use the optimizer cost estimates for modeling costs in the virtualization design problem. We only use the optimizer cost estimates to *rank* alternative execution plans, just as the optimizer normally does when choosing an execution plan for a query. This is important because the optimizer cost estimates are based on many simplifying assumptions, and so may not match the actual costs, but they can still be used to rank different alternatives.

We note that P for a given R depends only on the machine characteristics and does not change with the database or the query workload. Thus, we can obtain P for different R's off-line, and then use the different P values for all virtualization design problems, regardless of the database or query workload. This speeds up the process of using the optimizer for cost modeling, since the required P values for different R's are always readily available.

5. Calibrating the Optimizer

To obtain P for a given R, we use an experimental calibration process, which has to be performed only once for each R. In this process, we create a virtual machine on the physical machine that we will run our workloads on, and we run the database system inside this virtual machine. We set the level of resource allocation to this virtual machine as specified in R, and we run carefully designed synthetic queries on a synthetic database inside this virtual machine. The queries are designed so that measuring their actual execution time allows us to deduce the different values of parameters in P for this R. In effect, we are *calibrating* the query optimizer for the physical environment specified by the physical machine and R.

The query optimizers of some commercially available database systems have a built-in calibration process, and we can run the database system inside the virtual machine with resources set to R and simply invoke this calibration process. But many query optimizers do not have such a calibration process, and hence we must develop one by analyzing the query optimizer cost model and designing synthetic queries on a synthetic database so that the optimizer chooses specific plans for these queries and the cost of these plans is easy for the optimizer to estimate. We can then form equations that match the estimated costs obtained from the query optimizer cost formulas that have the parameters in P as variables with the actual, measured costs, and we can solve these equations to obtain P. It should be possible to use this method to obtain P for any database system, although the calibration process will vary depending on the details of the query optimizer cost formulas.

For our work, we use PostgreSQL as the database system, and we have designed an extensive set of calibration experiments based on the PostgreSQL cost model. As an example, consider two parameters in P, namely the `cpu_tuple_cost` and `cpu_operator_cost` parameters. These parameters are the query optimizer's measures of the CPU cost of processing each tuple and SQL `where` clause item (respectively) during a query, expressed as a fraction of the cost of a sequential page fetch. We can accurately calibrate these parameters by measuring the cost of the simple query `select max(R.a) from R`. If we ensure that there is no index on `R.a`, the optimizer will choose a plan that accesses `R` using a sequential scan. When executing this plan, the database system will pay a cost of `cpu_tuple_cost` for every tuple in `R`, in addition to another per tuple cost for the aggregation, `cpu_operator_cost`. The aggregation eliminates any overhead for returning the result, so the measured execution time of this query can be modeled as the weighted sum of these two per tuple costs. We can formulate another query whose execution time can be modeled as another equation in these two cost parameters, and we can simultaneously solve these two equations to obtain `cpu_tuple_cost` and `cpu_operator_cost`.

To illustrate this technique, we present the results of applying it in our experimental test bed. For our experiments, we use a machine with two 2.8GHz Intel Xeon CPUs (with hyperthreading) and 4GB of memory running SUSE Linux 10.0, kernel level 2.6.16. The database system we use is PostgreSQL 8.1.3. We use the Xen 3.0.2 hypervisor as our virtualization technology [23]. Xen enables us to vary the amount of memory allocated to a virtual machine during initial configuration and at run time. Xen also enables us to vary the amount of CPU allocated to a virtual machine by varying the scheduling time slice of this virtual machine at run time. Figure 3 shows the result of using our calibration process to compute `cpu_tuple_cost` for different CPU and memory allocations, ranging from 25% to 75% of the available CPU or memory. The figure shows that the `cpu_tuple_cost` parameter is sensitive to changes in resource allocation, and that our calibration process can detect this sensitivity.

6. An Illustrative Experiment

We illustrate the importance of virtualization design and the effectiveness of our cost modeling using the TPC-H benchmark. We use the OSDB implementation of the TPC-H benchmark [14], which includes an extensive set of indexes to boost performance. The database size is 1GB, and with indexes it is 4GB. The software and hardware configuration is as described in the last section.

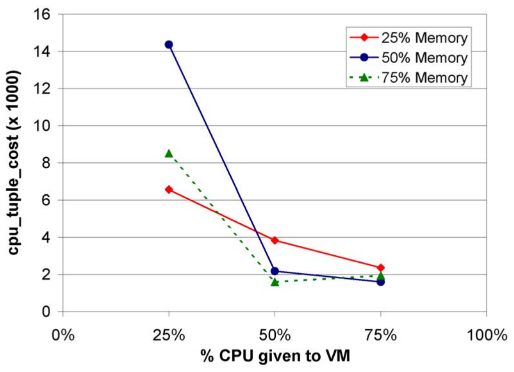

Figure 3. The `cpu_tuple_cost` parameter.

Consider two "workloads," one consisting of TPC-H Query 4 and the other consisting of TPC-H Query 13. If the two workloads will be run in virtual machines on the same physical machine, we need to determine the optimal resource allocation for each virtual machine. In this experiment, we focus on CPU and we give each virtual machine 50% of the available memory. The default CPU allocation would be to also give each virtual machine 50% of the available CPU. We will see next that this is not the best allocation.

We used the process described in Section 5 to calibrate the optimizer and obtain a cost model for a memory allocation of 50% of the available memory and CPU allocations of 25%, 50%, and 75% of the available CPU. Using these cost models, we estimated the execution time of Q4 and Q13. Figure 4 shows the estimated execution times, normalized to the estimated execution time of the default CPU allocation of 50%. The figure also shows the corresponding actual execution times measured for the queries, normalized to the actual execution time at the default CPU allocation. The estimated and actual execution times in the figure both show that Q4 is not sensitive to changing the CPU allocation. Most likely it is an I/O intensive query. On the other hand, Q13 is very sensitive to changing the CPU allocation. Thus, one possible virtualization design decision is to take CPU away from Q4 and give it to Q13. If we give 25% of the available CPU to Q4 and 75% to Q13, the performance of Q4 would not change significantly from that under the default CPU allocation of 50%, while the performance of Q13 would improve by a factor of two. This decision is made based on the estimated execution times shown in the figure, and the figure shows that the actual execution times validate this decision.

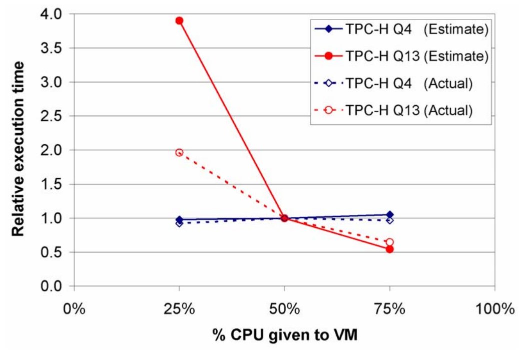

Figure 4. Sensitivity to varying CPU share.

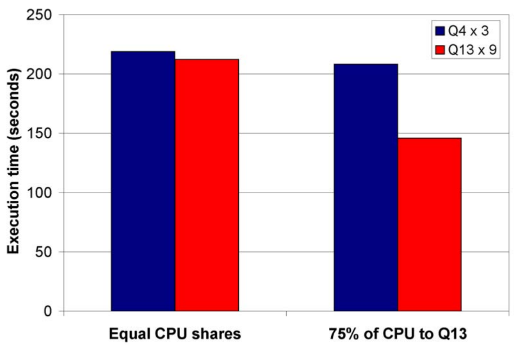

Figure 5. Effect on total execution time.

To test the effectiveness of allocating 25% of the available CPU to Q4 and 75% to Q13, we construct two workloads, one consisting of 3 copies of Q4 and the other consisting of 9 copies of Q13. We include multiple copies of the queries in our workload to reduce any effects of startup overheads, and we choose the number of copies of the queries so that the execution times of the two workloads are close to each other when they are each given equal shares of the CPU. Figure 5 shows the execution time of the workloads when they are each given an equal share of the CPU (default allocation), and when 75% of the CPU is given to Q13. The figure shows that the latter allocation improves the performance of Q13 by 30% without hurting the performance of Q4, and so is a better allocation than the default.

From this experiment we can see that (1) there can be better resource allocations than simply giving each workload an equal share of the available resources, and (2) our cost model is able to identify good resource allocations and can therefore be effectively used to solve the virtualization design problem.

7. Discussion and Next Steps

The cost modeling approach that we describe represents a key component to the solution of the virtualization design problem. This cost modeling can be refined by developing techniques to reduce the number of calibration experiments required, since cost model calibration is a fairly lengthy process. Speeding up cost model calibration may in itself be an interesting area for refinement.

As the optimizer is at the core of our modeling approach, we recognize that we inherit the well identified shortcomings of its cost modeling process (for instance, its weak modeling of the effect of concurrent query execution [11]). We can explore the application of more sophisticated techniques [4, 6, 8, 12] for modeling the relationship between resource allocation and query performance to the virtualization design problem.

Beyond the cost modeling, there remains the problem of developing a combinatorial search algorithm to search the space of possible resource allocations and find the best one. We envision this to be a straightforward task, and we believe that standard techniques such as dynamic programming will apply here.

An important next step in the area of tuning and virtualization, beyond the static virtualization design problem, is to consider the dynamic case and reconfigure the virtual machines on the fly in response to changes in the workload, database, number of virtual machines, or extra load on the physical resources. Adding different service-level objectives to the different workloads is also an interesting direction for future work.

Under our current approach, the database system need not be aware of the fact that it is running inside a virtual machine. We foresee that making database systems virtualization-aware, and allowing them to communicate with the virtualization layer, would enable a better configuration for both the virtual machine and the database system. The mechanisms for communication between the database system and the virtualization environment, and also the nature of the information exchange, are still open issues.

8. Conclusions

Resource virtualization can provide many benefits to database systems. But it also introduces new tuning and physical design problems that are of interest to researchers in self-managing database systems. In this paper, we discuss these problems, and we focus on one specific problem, namely *virtualization design*. We present a formulation of this problem and describe an approach to cost modeling, which is a key component in the solution to this problem. We also describe the next steps to solving this problem and present some directions for future work.

References

[1] A. Aboulnaga, C. Amza, and K. Salem. Virtualization and databases: State of the art and research challenges. In *Proc. Int. Conf. on Data Engineering (ICDE)*, 2007. (Advanced Technology Seminar).

[2] R. Agrawal, S. Chaudhuri, A. Das, and V. R. Narasayya. Automating layout of relational databases. In *Proc. Int. Conf. on Data Engineering (ICDE)*, 2003.

[3] P. T. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[4] M. J. Carey, R. Jauhari, and M. Livny. Priority in DBMS resource scheduling. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 1989.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, 2005.

[6] D. L. Davison and G. Graefe. Dynamic resource brokering for multi-user query execution. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1995.

[7] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood. Automatic performance diagnosis and tuning in Oracle. In *Proc. Conf. on Innovative Data Systems Research (CIDR)*, 2005.

[8] M. N. Garofalakis and Y. E. Ioannidis. Multi-dimensional resource scheduling for parallel queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1996.

[9] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. N. Tantawi. Dynamic placement for clustered web applications. In *Proc. Int. Conf. on World Wide Web (WWW)*, 2006.

[10] P. Martin, H.-Y. Li, M. Zheng, K. Romanufa, and W. Powley. Dynamic reconfiguration algorithm: Dynamically tuning multiple buffer pools. In *Proc. Int. Conf. Database and Expert Systems Applications (DEXA)*, 2000.

[11] D. T. McWherter, B. Schroeder, A. Ailamaki, and M. Harchol-Balter. Priority mechanisms for OLTP and transactional web applications. In *Proc. Int. Conf. on Data Engineering (ICDE)*, 2004.

[12] M. Mehta and D. J. DeWitt. Dynamic memory allocation for multiple-query workloads. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 1993.

[13] D. Narayanan, E. Thereska, and A. Ailamaki. Continuous resource monitoring for self-predicting DBMS. In *Proc. IEEE Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2005.

[14] The Open Source Database Benchmark. http://osdb.sourceforge.net/.

[15] M. Rosenblum and T. Garfinkel. Virtual machine monitors: Current technology and future trends. *IEEE Computer*, 38(5), 2005.

[16] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *Proc. Symposium on Operating System Design and Implementation (OSDI)*, 2002.

[17] J. E. Smith and R. Nair. The architecture of virtual machines. *IEEE Computer*, 38(5), 2005.

[18] A. J. Storm, C. Garcia-Arellano, S. Lightstone, Y. Diao, and M. Surendra. Adaptive self-tuning memory in DB2. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 2006.

[19] Virtual Appliances. http://www.virtualappliances.net/.

[20] Virtual Iron. http://www.virtualiron.com/.

[21] VMware. http://www.vmware.com/.

[22] G. Weikum, A. Mönkeberg, C. Hasse, and P. Zabback. Self-tuning database technology and information services: from wishful thinking to viable engineering. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 2002.

[23] XenSource. http://www.xensource.com/.