Deploying Database Appliances in the Cloud

Ashraf Aboulnaga* Kenneth Salem* Ahmed A. Soror* Umar Farooq Minhas*
Peter Kokosielis† Sunil Kamath†

*University of Waterloo
†IBM Toronto Lab

Abstract

Cloud computing is an increasingly popular paradigm for accessing computing resources. A popular class of computing clouds is Infrastructure as a Service (IaaS) clouds, exemplified by Amazon's Elastic Computing Cloud (EC2). In these clouds, users are given access to virtual machines on which they can install and run arbitrary software, including database systems. Users can also deploy database appliances on these clouds, which are virtual machines with pre-installed pre-configured database systems. Deploying database appliances on IaaS clouds and performance tuning and optimization in this environment introduce some interesting research challenges. In this paper, we present some of these challenges, and we outline the tools and techniques required to address them. We present an end-to-end solution to one tuning problem in this environment, namely partitioning the CPU capacity of a physical machine among multiple database appliances running on this machine. We also outline possible future research directions in this area.

1 Introduction

Cloud computing has emerged as a powerful and cost-effective paradigm for provisioning computing power to users. In the cloud computing paradigm, users use an intranet or the Internet to access a shared computing cloud that consists of a large number (thousands or tens of thousands) of interconnected machines organized as one or more clusters. This provides significant benefits both to providers of computing power and to users of this computing power. For providers of computing power, the push to cloud computing is driven by economies of scale. By operating massive clusters in specially designed and carefully located data centers, providers can reduce administrative and operating costs, such as the costs of power and cooling [15, 16]. In addition, the per-unit costs of hardware, software and networking become significantly cheaper at this scale [4]. For users, cloud computing offers simple and flexible resource provisioning without up-front equipment and set up costs and on-going administrative and maintenance burdens. Users can run software in the cloud, and they can grow and shrink the computing power available to this software in response to growing and shrinking load [4].

There are different flavors of cloud computing, depending on how much flexibility the user has to customize the software running in the cloud. In this paper, we focus on computing clouds where the user sees a bare-bones machine with just an operating system and gets full flexibility in installing and configuring software on this machine. These clouds are known as *Infrastructure as a Service (IaaS)* clouds. A very prominent example

Copyright 2009 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

of this type of cloud is Amazon's Elastic Computing Cloud (EC2) [2], which enables users to rent computing power from Amazon to run their software. Other providers of this style of cloud computing include GoGrid [13] and AppNexus [3]. Additionally, many organizations are building IaaS clouds for their internal use [6, 22].

In IaaS clouds, users are typically given access to *virtual machines (VMs)* [5, 23] on which they can install and run software. These virtual machines are created and managed by a *virtual machine monitor (VMM)* which is a layer of software between the operating system and the physical machine. The VMM controls the resources of the physical machine, and can create multiple VMs that share these physical machine resources. The VMs have independent operating systems running independent applications, and are isolated from each other by the VMM. The VMM controls the allocation of physical machine resources to the different VMs. The VMM also provides functionality such as saving and restoring the image of a running VM, or migrating VMs between physical machines.

A common model for deploying software in virtual machine environments is the *virtual appliance* model. A virtual appliance is a VM image with a pre-installed pre-configured application. Deploying the application simply requires copying this VM image to a physical machine, starting the VM, and performing any required configuration tasks. The cost of installing and configuring the application on the VM is incurred once, when the appliance is created, and does not need to be incurred again by users of the appliance. A *database appliance* is a virtual appliance where the installed application is a database system. With the increasing popularity of virtualization and cloud computing, we can expect that a common way of providing database services in the future will be through **database appliances deployed in IaaS clouds**. As an example of this deployment mode, Amazon offers MySQL, Oracle, and Microsoft SQL Server virtual appliances for deployment in its EC2 cloud.

An important question to ask is how to get the best database system performance in this environment. Cloud providers are interested in two related performance objectives: maximizing the utilization of cloud resources and minimizing the resources required to satisfy user demand. Users are interested in minimizing application response time or maximizing application throughput. Deploying database appliances in the cloud and tuning the database and virtualization parameters to optimize performance introduces some interesting research challenges. In this paper, we outline some of these challenges (Section 2), and we present the different tools and techniques required to address them (Section 3). We present our work on partitioning CPU capacity among database appliances as an example end-to-end tuning solution for virtualized environments (Section 4). We conclude by outlining some possible future research directions in this area (Section 5).

2 Deployment and Tuning Challenges

Our focus is on deploying and tuning virtual machines running database systems (i.e., database appliances) on large clusters of physical machines (i.e., computing clouds). This raises deployment and computing challenges, which we describe next.

2.1 Deployment Challenges

Creating a database appliance that can easily be deployed in a cloud, and obtaining an accessible, usable database instance from this appliance require addressing many issues related to deployment. These issues are not the research focus of our work, but we present them here since these seemingly simple and mundane tasks can be very tricky and time consuming. These issues include:

Localization:
When we start a VM from a copy of a database appliance, we need to give this new VM and the database system running on it a distinct "identity." We refer to this process as *localization*. For example, we need to give the VM a MAC address, an IP address, and a host name. We also need to adapt (or localize) the database instance running on this VM to the VM's new identity. For example, some database systems require every database

instance to have a unique name, which is sometimes based on the host name or IP address. The VMM and the underlying operating system and networking infrastructure may help with issues such as assigning IP addresses, but there is typically little support for localizing the database instance. The specific localization required varies from database system to database system, which increases the effort required for creating database appliances.

Routing:
In addition to giving every VM and database instance a distinct identity, we must be able to route application requests to the VM and database instance. This includes the IP-level routing of packets to the VM, but it also includes making sure that database requests are routed to the correct port and not blocked by any firewall, that the display is routed back to the client console if needed, that I/O requests are routed to the correct virtual storage device if the "compute" machines of the IaaS cloud are different from the storage machines, and so on.

Authentication:
The VM must be aware of the credentials of all clients that need to connect to it, independent of where it is run in the cloud.

2.2 Tuning Challenges

Next, we turn our attention to the challenges related to tuning the parameters of the virtualization environment and the database appliance to achieve the desired performance objectives. These are the primary focus of our research work, and they include:

Placement:
Virtualization allows the cloud provider to run a user's VM on any available physical machine. The mapping of virtual machines to physical machines can have a significant impact on performance. One simple problem is to decide how many virtual machines to run on each physical machine. The cloud provider would like to minimize the number of physical machines used, but running more VMs on a physical machine degrades the performance of these VMs. It is important to balance these conflicting objectives: minimizing the number of physical machines used while maintaining acceptable performance for users.

A more sophisticated mapping of virtual machines to physical machines could consider not only the number of VMs per physical machine, but also the resource requirements of these VMs. The placement algorithm could, for example, avoid mapping multiple I/O intensive VMs to the same physical machine to minimize I/O interference between these VMs. This type of mapping requires understanding the resource usage characteristics of the application running in the VM, which may be easier to do for database systems than for other types of applications since database systems have a highly stylized and often predictable resource usage pattern.

Resource Partitioning:
Another tuning challenge is to decide how to partition the resources of each physical machine among the virtual machines that are running on it. Most VMMs provide tools or APIs for controlling the way that physical resources are allocated. For example VMM scheduling parameters can be used to apportion the total physical CPU capacity among the VMs, or to control how virtual CPUs are mapped to physical CPUs. Other VMM parameters can be used to control the amount of physical memory that is available to each VM. To obtain the best performance, it is useful to take into account the characteristics of the application running in the VM so that we can allocate resources where they will provide the maximum benefit. Database systems can benefit from this application-informed resource partitioning, as we will show in Section 4.

Service Level Objectives:
To optimize the performance of a database appliance in a cloud environment, it is helpful to be able to express different *service level objectives*. The high-level tuning goal is to minimize the cloud resources required while maintaining adequate performance for the database appliance. Expressing this notion of "adequate performance" is not a trivial task. A database system is typically part of a multi-layer software stack that is used to

serve application requests. Service level agreements are typically expressed in terms of end-to-end application performance, with no indication of how much of this performance budget is available to the database system vs. how much is available to other layers of the software stack (e.g., the application server and the web server). Deriving the performance budget that is available to the database system for a given application request is not easy, since an application request can result in a varying number of database requests, and these database requests can vary greatly in complexity depending on the SQL statements being executed. Tuning in a cloud environment therefore requires developing practical and intuitive ways of expressing database service level objectives. Different workloads can have different service level objectives, and the tuning algorithms need to take these different service level objectives into account.

Dynamically Varying Workloads:
Tuning the performance of a database appliance (e.g., placement and resource partitioning) requires knowledge of the appliance's workload. The workload can simply be the full set of SQL statements that execute at the appliance. However, it is an interesting question whether there can be a more succinct but still useful representation of the workload. Another interesting question is whether some tuning decisions can be made without knowledge of the SQL statements (e.g., if this is a new database instance). It is also important to detect when the nature of the workload has changed, possibly by classifying the workload [11] and detecting when the workload class has changed. The tuning algorithms need to be able to deal with dynamically changing workloads that have different service level objectives.

3 Tools and Techniques

Next, we turn our attention to the tools and techniques that are needed to address the tuning challenges outlined above. These include:

Performance Models:
Predicting the effect of different tuning actions on the performance of a database appliance is an essential component of any tuning solution. This requires developing accurate and efficient performance models for database systems in virtualized environments. There are two general classes of models: white box models, which are based on internal knowledge of the database system, and black box models, which are typically statistical models based on external, empirical observations of the database system's performance.

White box modeling is especially attractive for database systems for two reasons. First, database systems have a stylized and constrained interface for user requests: they accept and execute SQL statements. This simplifies defining the inputs to the performance model. Second, and more importantly, database systems already have highly refined internal models of performance. One way to build a white box model is to expose these internal models to the tuning algorithm and adapt them to the tuning task at hand. For example, the query optimizer cost model, which has been used extensively as a what-if cost model for automatic physical database design [7], can be used to quantify the effect of allocating different shares of physical resources to a database appliance (see the next section for more details). Self-managing database systems have other internal models that can be exposed for use in performance tuning in a cloud environment. These include the memory consumption model used by a self-tuning memory manager [8, 21] or the model used for automatic diagnosis of performance problems [10].

The disadvantage of white box modeling is that the required performance models do not always exist in the database system, and developing white box models from scratch is difficult and time consuming. Even when internal models do exist in the database system, these models are sometimes not calibrated to accurately provide the required performance metric, and they sometimes make simplifying assumptions that ignore important aspects of the problem. For example, the query optimizer cost model is designed primarily to compare query execution plans, not to accurately estimate resource consumption. This cost model focuses on one query at a time, ignoring the sometimes significant effect of concurrently running interacting queries [1]. Because of these shortcomings of white box modeling, it is sometimes desirable to build black box models of performance by fitting statistical models to the observed results of performance experiments [1]. When building these models it is important to carefully decide which performance experiments to conduct to collect samples for the model, since these experiments can be costly and they have a considerable impact on model accuracy [18]. However, the illusion of infinite computing resources provided by IaaS clouds can actually simplify black box experimental modeling of database systems, since we can now easily provision as many machines as we need to run the performance experiments required for building an accurate model.

An interesting research question is whether it is possible to combine the best features of black box and white box modeling, by using the internal models of the database system as a starting point, but then refining these models based on experimental observations [12].

Optimization and Control Algorithms:
Solving the performance tuning problems of a cloud environment requires developing combinatorial optimization or automatic control algorithms that use the performance models described above to decide on the best tuning action. These algorithms can be static algorithms that assume a fixed workload, or they can be dynamic algorithms that adapt to changing workloads. The algorithms can simply have as a goal the best-effort maximization of performance [20], or they can aim to satisfy different service level objectives for different workloads [17].

Tools for System Administrators:
In addition to the models and algorithms described above, system administrators need tools for deploying and tuning database appliances. These tools should expose not only the performance characteristics of the VM, but also the performance characteristics of the database system running on this VM. For example, it would be useful to expose the what-if performance models of the database system to system administrators so that they can make informed tuning decisions, diagnose performance problems, or refine the recommendations of automatic tuning algorithms.

Co-tuning and Hint Passing:
The focus of the previous discussion has been on tuning virtual machine parameters. It is also important to tune the parameters of the database system running on this virtual machine. For example, if we decide to decrease the memory available to a VM running a database system, we need to decrease the sizes of the different memory pools of this database system. This *co-tuning* of VM and database system parameters is important to ensure that the tuning actions at one layer are coordinated with the tuning action at the other layer. Another way to coordinate VM tuning with database system tuning is to pass *hints* that can be used for tuning from the database system to the virtualization layer. These hints would contain information that is easy to obtain for the database system and useful for tuning at the virtualization layer. For example, these hints could be used to ensure that VM disks storing database objects (i.e., tables or indexes) that are accessed together are not mapped to the same physical disk. Information about which objects are accessed together is easily available to the database system and very useful to the virtualization layer.

4 Virtual Machine Configuration

In this section, we consider the following tuning problem: Given N virtual machines that share one physical machine, with each VM running an independent database system instance, how can we optimally partition the available CPU capacity of the physical machine among the virtual machines? Recall that the VMM provides mechanisms for deciding how much CPU capacity is allocated to each VM. We outline a solution to this resource partitioning problem below. Full details of our solution can be found in [19, 20].

We decide the partitioning of the available CPU capacity of the physical machine among the N virtual machines with the goal of maximizing the aggregate throughput of the N workloads (or minimizing their total completion time). This is a best-effort performance goal that does not consider explicit service level objectives for the different workloads.

The benefit that each database system will obtain from an increase in CPU allocation depends on that system's workload. We assume that we are given the set of SQL statements that make up the workload of each of the N database systems. These workloads represent the SQL statements executed by the different database systems in the same time interval, so the number of statements in a workload corresponds to its intensity (i.e., the rate of arrival of SQL statements). We assume that the workloads are fixed, and we do not deal with dynamically varying workloads.

To determine the best CPU partitioning, we need a model of the performance of a database workload as a function of the CPU capacity allocated to the VM running this workload. In our solution, we use the cost model of the database system's query optimizer as a what-if model to predict performance under different CPU allocations. This requires the query optimizer cost model to be aware of the effect of changing CPU capacity on performance. The cost model relies on one or more modeling parameters to describe CPU capacity and estimate the CPU cost of a query. We use different values of these CPU modeling parameters for different CPU allocations, thereby adding awareness of CPU allocation to the query optimizer cost model. We call such a cost model *virtualization aware*. The calibration procedure required to determine the values of the CPU modeling parameters to use for each CPU allocation is performed only once for every database system and physical machine configuration, and can be used for any workload that runs on this database system.

We use the virtualization aware cost models of the N database systems on the N VMs in a greedy search algorithm to determine the best partitioning of CPU capacity among the VMs. We also provide heuristics for refining the cost models based on comparing estimated performance to actual observed performance. We apply these refinement heuristics periodically, and we obtain a new partitioning of CPU capacity after each refinement of the cost model.

To illustrate the effectiveness of our approach, consider the following example (Figure 1). Using the Xen VMM [5] we created two virtual machines, each running an instance of PostgreSQL. We ran both VMs on the same physical machine, a Sun server with two 2.2GHz dual core AMD Opteron Model 275 x64 processors and 8GB memory, running SUSE Linux 10.1. For this example, we used a TPC-H database with scale factor 1. On one PostgreSQL instance we ran a workload consisting of three instances of TPC-H query $Q4$. On the other instance, we ran a workload consisting of nine instances of TPC-H query $Q13$. First, we allocated 50% of the available CPU capacity to each of the two virtual machines, ran the two workloads, and measured the total execution time of each workload. The results are illustrated by the bars on the left for each of the two workloads in Figure 1. Next, we repeated the experiment, but this time we allocated CPU capacity according to the recommendations of our CPU partitioning algorithm. The algorithm recommended giving 25% of the available CPU capacity to the first PostgreSQL instance (Workload 1) and the remaining 75% to the second instance (Workload 2). The execution times of the two workloads under this CPU allocation are shown in Figure 1 by the bars on the right for each of the two workloads. This change in CPU allocation reduces the execution time of the second workload by approximately 30%, while having little impact on the first workload. Thus, we can see the importance of correctly partitioning CPU capacity and the effectiveness of our approach to solving this problem.

5 Future Directions

The previous section illustrates a simple performance tuning problem in a cloud computing environment and its solution. Extending the research outlined in the previous section opens up many possibilities for future work, which we are exploring in our ongoing research activities. Instead of partitioning the resources of one physical machine among the VMs, we can consider *multiple* physical machines and partition their resources among the VMs, that is, decide which physical machine to use for each VM and what share of this machine's resources are

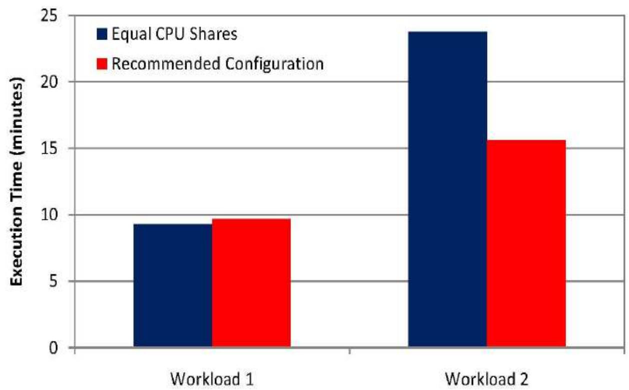

Figure 1: Effect of varying CPU allocation on workload performance.

allocated to the VM. We can also extend the work to deal with dynamically varying workloads, possibly with different explicit service level objectives. Another interesting research direction is improving the way we refine the query-optimizer-based cost model in response to observed performance.

Another interesting research direction is optimizing the allocation of I/O resources to different VMs. Some VMMs, such as VMWare ESX server [23], provide mechanisms for controlling how much of the I/O bandwidth of a physical machine is allocated to each VM running on this machine. Another mechanism to control the allocation of I/O resources to VMs is controlling the mapping of VM disks to physical disks. Using these two mechanisms to optimize the performance of database appliances is an interesting research direction, especially since many database workloads are I/O bound.

It would also be interesting to explore whether we can expose internal database system models other than the query optimizer cost model and use these models for tuning VM parameters or co-tuning VM and database system parameters. For example, the memory manager performance model can be used to control memory allocation.

The cloud environment also offers new opportunities, beyond the challenges of tuning database appliances. For example, since we can provision VMs on-demand, it would be interesting to explore the possibility of scaling out a database system to handle spikes in the workload by starting new replicas of this database system on newly provisioned VMs. This requires ensuring consistent access to the database during and after the replication process, coordinating request routing to the old and new VMs, and developing policies for when to provision and de-provision new replicas.

Finally, this idea of application-informed tuning of the virtualized environment is not restricted to database systems. This idea can be used for other types of applications that run in a cloud environment, such as large scale data analysis programs running on Map-Reduce style platforms [9, 14].

6 Conclusion

As cloud computing becomes more popular as a resource provisioning paradigm, we will increasingly see database systems being deployed as virtual appliances on Infrastructure as a Service (IaaS) clouds such as Amazon's EC2. In this paper, we outlined some of the challenges associated with deploying these appliances and tuning their performance, and we discussed the tools and techniques required to address these challenges. We presented an end-to-end solution to one tuning problem, namely partitioning the CPU capacity of a physical machine among the database appliances running on this machine. We also described some future directions for this research area. It is our belief that the style of application-informed tuning described in this paper can provide significant benefits to both providers and users of cloud computing.

References

[1] Mumtaz Ahmad, Ashraf Aboulnaga, Shivnath Babu, and Kamesh Munagala. Modeling and exploiting query interactions in database systems. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, 2008.

[2] Amazon EC2. http://aws.amazon.com/ec2/.

[3] AppNexus. http://www.appnexus.com/.

[4] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, EECS Department, University of California, Berkeley, Feb 2009.

[5] Paul T. Barham, Boris Dragovic, Keir Fraser, Steven Hand, Timothy L. Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proc. ACM Symp. on Operating Systems Principles (SOSP)*, 2003.

[6] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, Jan/Feb 2003.

[7] Surajit Chaudhuri and Vivek R. Narasayya. An efficient cost-driven index selection tool for Microsoft SQL Server. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 1997.

[8] Benoît Dageville and Mohamed Zaït. SQL memory management in Oracle9i. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 2002.

[9] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. Symp. on Operating System Design and Implementation (OSDI)*, 2004.

[10] Karl Dias, Mark Ramacher, Uri Shaft, Venkateshwaran Venkataramani, and Graham Wood. Automatic performance diagnosis and tuning in Oracle. In *Proc. Conf. on Innovative Data Systems Research (CIDR)*, 2005.

[11] Said Elnaffar, Patrick Martin, and Randy Horman. Automatically classifying database workloads. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, 2002.

[12] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet Wiener, Armando Fox, Michael Jordan, and David Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, 2009.

[13] GoGrid. http://www.gogrid.com/.

[14] Hadoop. http://hadoop.apache.org/.

[15] James R. Hamilton. Cost of power in large-scale data centers, Nov 2008. http://perspectives.mvdirona.com/2008/11/28/CostOfPowerInLargeScaleDataCenters.aspx.

[16] Randy H. Katz. Tech titans building boom. *IEEE Spectrum*, Feb 2009.

[17] 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 *Proc. European Conf. on Computer Systems (EuroSys)*, 2007.

[18] Piyush Shivam, Varun Marupadi, Jeffrey S. Chase, Thileepan Subramaniam, and Shivnath Babu. Cutting corners: Workbench automation for server benchmarking. In *Proc. USENIX Annual Technical Conference*, 2008.

[19] Ahmed A. Soror, Ashraf Aboulnaga, and Kenneth Salem. Database virtualization: A new frontier for database tuning and physical design. In *Proc. Workshop on Self-Managing Database Systems (SMDB)*, 2007.

[20] Ahmed A. Soror, Umar Farooq Minhas, Ashraf Aboulnaga, Kenneth Salem, Peter Kokosielis, and Sunil Kamath. Automatic virtual machine configuration for database workloads. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2008.

[21] Adam J. Storm, Christian Garcia-Arellano, Sam Lightstone, Yixin Diao, and Maheswaran Surendra. Adaptive self-tuning memory in DB2. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 2006.

[22] Virtual Computing Lab. http://vcl.ncsu.edu/.

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