A Demonstration of SQLVM: Performance Isolation in Multi-Tenant Relational Database-as-a-Service

Vivek Narasayya[§] Sudipto Das[§] Manoj Syamala[§]
Surajit Chauduri[§] Feng Li[†] Hyunjung Park[‡]

[§]Microsoft Research
Redmond, WA 98052, USA
{viveknar, sudiptod, manojsy, surajitc}@microsoft.com

[†]NUS
Singapore 117590
li-feng@comp.nus.edu.sg

[‡]Stanford University
Stanford, CA 94305, USA
hyunjung@stanford.edu

ABSTRACT

Sharing resources of a single database server among multiple tenants is common in *multi-tenant* Database-as-a-Service providers, such as Microsoft SQL Azure. Multi-tenancy enables cost reduction for the cloud service provider which it can pass on as savings to the tenants. However, resource sharing can adversely affect a tenant's performance due to other tenants' workloads contending for shared resources. Service providers today do not provide any assurances to a tenant in terms of isolating its performance from other co-located tenants. SQLVM, a project at Microsoft Research, is an abstraction for performance isolation which is built on a promise of reserving key database server resources, such as CPU, I/O and memory, for each tenant. The key challenge is in supporting this abstraction within a RDBMS without statically allocating resources to tenants, while ensuring low overheads and scaling to large numbers of tenants. This demonstration will show how SQLVM can effectively isolate a tenant's performance from other tenant workloads co-located at the same database server. Our demonstration will use various scripted scenarios and a data collection and visualization framework to illustrate performance isolation using SQLVM.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Relational databases*

General Terms

Design, Experimentation, Measurement, Performance

Keywords

Cloud computing, multitenancy, performance isolation, resource isolation, relational database-as-a-service

1. INTRODUCTION

Services such as Microsoft SQL Azure, which offer relational Database-as-a-Service (DaaS) functionality in the cloud, are designed to be *multi-tenant*; a single database server process hosts databases of different tenants. Figure 1 illustrates such a multi-tenant relational database (*RDBMS*) architecture. Multi-tenancy is crucial to make the service economical since dedicating a machine for each tenant is expensive. Such multi-tenancy in DaaS is also relevant for on-premise clouds where a server consolidates databases of multiple independent applications within an enterprise.

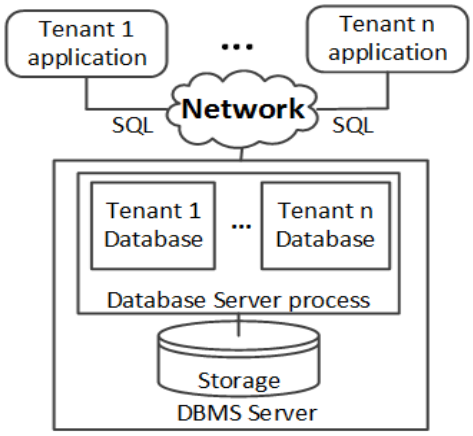


Figure 1: A multi-tenant relational database system where multiple tenants share the same database process and content for resources at the server.

A consequence of multi-tenancy and resource sharing is that a tenant's workload competes with workload from *other* co-located tenants for *key resources* such as CPU, I/O and memory of the database server. As a result, the performance of a tenant's workload can vary significantly depending on the workloads issued concurrently by other tenants. Such performance unpredictability arising from contention with other tenants for shared database server resources is a serious concern for performance sensitive tenants.

While *assurances* on high-level performance metrics at the level of SQL queries, e.g., throughput (queries/sec) or query latency, is desirable, it is also extremely challenging to provide such guarantees for arbitrary SQL queries which the tenants execute. Even on a database server that is *exclusively* used by one tenant, the resource needs and execution times of different instances of a single query template, such as a stored procedure, can vary dramatically depending on parameter values. In addition, service providers need to also support ad-hoc queries (i.e., queries not seen previously) without limiting the workload type or the SQL query language supported. Furthermore, a tenant's data size, distribution, and access patterns can change over time. These factors contribute to even greater variability in throughput and latency.

SIGMOD'13, June 22–27, 2013, New York, New York, USA.

A fundamental challenge, however, is to reduce the variability in performance that arises due to contention with other tenants for critical shared database server resources. That is, a tenant's performance must be unaffected by the workloads of other co-located tenants. Our approach is to provide the tenants assurances at the level of key resources such as CPU, I/O, buffer pool memory, and working memory for operators such as hash and sort. The rationale is that isolating the key resources will provide performance isolation which can be leveraged to support higher-level assurances.

At first glance, it may appear that techniques developed for resource management in traditional enterprise DBMS may be adequate for such resource-level assurances. These techniques are typically based on relative priorities, proportional sharing, or enforcing maximum limits on resource usage. However, particularly in a public cloud setting, a major drawback of relative priorities and proportional sharing is that the assurance of how much resources a tenant will receive is not *absolute* and depends on which other tenants are active (and their priorities/shares). Similarly, enforcing maximum limits also suffers from this drawback when the service provider *overbooks* (i.e., promise more resources in aggregate to tenants than the available system capacity) to increase utilization and cost-effectiveness. In contrast, resource reservations in SQLVM is much more meaningful to a tenant, since the assurance of how much resources the tenant will receive is independent of other tenants. As a concrete example, consider the I/O resource. Suppose tenant T_1 is promised a reservation of 100 I/Os per second (IOPS). The promise is that if T_1's workload demands 100 IOPS (or more) and the system assumes responsibility for granting 100 IOPS no matter which other tenants are executing concurrently on the server.

In order to be cost-effective and support high consolidation factors, static resource allocation to support such resource reservations is impractical. It is therefore possible that a tenant may not always receive the resources it was promised (e.g., due to overbooking). Thus, *metering* the promised reservation becomes crucial to establish accountability. That is, if resource allocation is less than the reservation, one can objectively determine if the tenant's workload had less demand for resources (i.e., no violation), or if the system did not allocate resources even though there was sufficient demand (i.e., a violation). Referring to the I/O example above, suppose T_1 was allocated 80 IOPS when its reservation is 100 IOPS. If T_1's queries did not generate sufficient I/O requests, the provider did not violate the promise. On the other hand, if T_1 generated sufficient I/O requests, T_1's I/O reservation was violated. The challenge of metering is to determine sufficient demand for each resource without making *any* assumptions about the workload or limiting the workload. Metering is independent of the actual resource allocation mechanisms and is essential to establish trust between the tenant and the provider who might have conflicting interests.

SQLVM [4], a project at Microsoft Research, *is a reservation of a set of key resources for a tenant inside the database server*. Reservations in SQLVM is a promise of a minimum amount of resources and every resource promise has an associated metering logic. Conceptually, the tenant is exposed a familiar abstraction of a virtual machine (*VM*) with a specified set of resources such as CPU, I/O and memory, but inside the database server. SQLVM's assurances apply to *any* RDBMS workload without any restrictions, assumptions, or limitations. Internally, new promise-aware resource allocation mechanisms exercise fine-grained control to orchestrate shared resources across tenants without requiring static allocation upfront. If a tenant's resource reservation is not met, then metering logic for that resource establishes accountability. Note that the obvious alternative of actually creating VMs (one per tenant) and running an instance of the database server process within each VM is too heavyweight and fails to achieve the degree of consolidation demanded for DaaS [1]. In contrast, SQLVM is much more lightweight, allowing consolidation factors of hundreds of tenants.

Curino et al. [1] and Lang et al. [3] approach consolidation of multiple databases in a single server by analyzing the workloads, identifying how these workloads interact with one another, and recommending which databases should be co-located in order to meet performance goals (or Service-Level-Objectives). Xiong et al. [5] constructs machine learning models to predict query performance as a function of resources allocated to it, and then exploit such models to allocate resources so that query latency SLO can be met. SQLVM is complementary to these approaches since it provides resource-level isolation for tenants and makes no assumptions about the specific workloads of tenants. SQLVM can potentially be used as a building block to build such recommenders, since SQLVM can ensure that the tenants are actually allocated the resources that the models assume. There has also been extensive work in area of workload management, particularly in a traditional data warehouse setting where queries can be resource intensive (Krompass et al. [2] present an overview). We believe that SQLVM can be valuable even in such traditional enterprise scenarios since it can be used to more tightly control resource allocation to different classes of queries. Finally, resource reservation and scheduling mechanisms have been explored both in the operating systems and virtualization contexts. However, DBMSs typically need to assume control of most systems resources, and therefore cannot benefit directly from such OS or hypervisor level mechanisms. Furthermore, database workloads and the DaaS context bring unique challenges that require us to rethink the assurances and the mechanisms necessary to support them within the DBMS.

Narasayya et al. [4] presented the detailed SQLVM model, the challenges in efficiently implementing this model within a commercial RDBMS engine, and outlined some of resource scheduling techniques used to support the model. This demonstration will highlight the benefits of the SQLVM abstraction from the tenant's perspective. In particular, we will focus SQLVM's ability to provide performance isolation while other tenants are being added or removed from the system. We will demonstrate how a tenant within a SQLVM with resource reservations is unaffected as other co-located tenant workloads are added or removed. We will use a mix of workloads to emulate the variety likely to be observed in a multi-tenant platform. With the help of a data collection and visualization framework, the audience will be able to monitor and visualize each tenant's end-to-end performance and the resource utilization.

2. SQLVM DESIGN OVERVIEW

SQLVM is a reservation of a set of key resources in a database system, such as CPU, I/O, and memory. Conceptually, a tenant is promised a VM with specified resources, but within the database server process. Unlike a traditional VM, a SQLVM is much more lightweight since its only goal is to provide resource isolation across tenants. Reservations in a SQLVM is a promise of a minimum amount of resources which will be allocated if the tenant has sufficient demand for the resources. In this section, we provide a high-level overview of the SQLVM model; see [4] for more details about the model and the I/O scheduling mechanism.

2.1 CPU

For a tenant, and a given core, the CPU utilization over an interval of time is defined as the percentage of time for which a task of that tenant is running on that core. This definition extends naturally

to the case of k cores as the total time for which tasks of that tenant run across all cores, as a percentage of ($k\times$ time interval).

Promise: SQLVM promises to reserve for the tenant (T_i) a CPU utilization, denoted by $ResCPU_i$, equivalent to a slice of the CPU time on available core(s). For example, if $ResCPU = 10\%$, then in a metering interval of 1 sec, the tenant should be allocated CPU time of at least 100 msec, provided the tenant has sufficient work. CPU reservations can correspond to a fraction of a core, thus allowing better consolidation ratios since reservations can be supported to many more tenants than available cores.

Metering: The metering problem is as follows: of the total time during which T_i had at least one task running or ready to run, it must receive at least $ResCPU_i$ percentage of the CPU; the provider violated the promise otherwise. For instance, if T_1 was promised $ResCPU_1 = 10\%$ and if T_1 had at least one task ready to run (or running) for 500 ms, the provider violates the promise only if the allocated CPU is less than 50 ms, i.e., T_1's effective utilization is less than 10%. This definition of metering is fair since the provider is not held accountable for the tenant being idle (i.e., no tasks ready to run), while ensuring that a provider cannot arbitrarily delay a tenant's task without violating the promise.

2.2 I/O

Fine-grained sharing of the disk bandwidth is important. For simplicity in the discussion below we refer to I/O throughput, although the definitions can be extended for bandwidth (such as bytes per second) as well.

Promise: SQLVM promises to reserve for the tenant a certain IOPS, denoted $ResIOPS_i$. This promise can again be viewed as a slice of the IOPS capacity available of the underlying physical disk drives. Note that our promise makes no distinction between sequential and random I/Os.

Metering: The key challenge in metering I/O throughput is in determining if the tenant had "sufficient I/O requests" to meet its reservation and whether the I/O throughput achieved is commensurate with the promise. Observe that if a tenant had at least one I/O request pending, then it had work to utilize the I/O resources. We define the effective I/O throughput achieved as the IOPS achieved for the time when the tenant had at least one pending I/O request in the given metering interval. The I/O metering logic flags a violation if the effective I/O throughput is less than $ResIOPS_i$.

2.3 Memory

While there are many uses of memory in a relational DBMS, we focus here on the two major uses: buffer pool and working memory. The buffer pool is a cache of database pages that is managed using a page replacement strategy (e.g., LRU-k). If a page is not found in the buffer pool, the DBMS incurs I/O to obtain it from secondary storage. Working memory is private to a physical operator used in a query execution plan, such as Hash or Sort. If working memory is limited, the operator may need to spill its state (e.g., partitions of the hash table) to secondary storage, thus again incurring additional I/O. Therefore, promises on memory are also crucial for performance. Similar to static reservation of CPU and I/O capacity, statically allocating a tenant's memory also limits consolidation. Therefore, SQLVM dynamically distributes memory across tenants, but provides a precise promise to tenants that exposes an illusion of statically-allocated memory.

Promise: To allow dynamic and fine-grained sharing of memory among tenants, our promise is that the number of I/Os incurred in the multi-tenant system is the same as though the system had dedicated a certain amount (say 1 GB) of buffer pool memory for the tenant; a similar promise applies for working memory. For a given amount of memory M, we define Relative IO as follows:

$$\textit{Relative IO} = \frac{\textit{Actual IOs} - \textit{Baseline IOs (M)}}{\textit{Baseline IOs (M)}} \quad (1)$$

SQLVM promises a tenant Relative IO ≤ 0 for a given amount of memory. Similar to other resources, a tenant is promised a memory reservation ($ResMem_i$). For example, suppose a tenant is promised a 1 GB buffer pool memory reservation. In effect, the promise is that the tenant's workload will see the same hit ratio as though a 1 GB buffer pool was reserved for the tenant. Similarly for working memory, a promise of 500 MB implies that there would be no more I/O to/from disk for Hash or Sort operators compared to 500 MB of working memory dedicated to that tenant.

Metering: Since memory is allocated dynamically and a tenant's actual memory allocation might differ from $ResMem_i$, the key challenge for metering memory is to determine Baseline IOs (M); Actual IOs can be measured directly. SQLVM simulates the I/O behavior of the workload as though the tenant had M units of memory dedicated to it. The challenge lies in doing this baseline simulation accurately and with low overhead. For example, for buffer pool memory, the observation is that the relative I/O is dependent on the page access order, page replacement policy and page metadata (such as dirty bit), and not the actual contents of the pages. The CPU overhead to simulate this baseline buffer pool can be piggybacked on the actual buffer pool accesses and page replacement, and is almost negligible in practice. Finally, any I/O incurred as a result of insufficient memory allocation, as flagged by the metering logic, is not charged against the tenant's I/O reservation ($ResIOPS_i$).

3. DEMONSTRATION DESCRIPTION

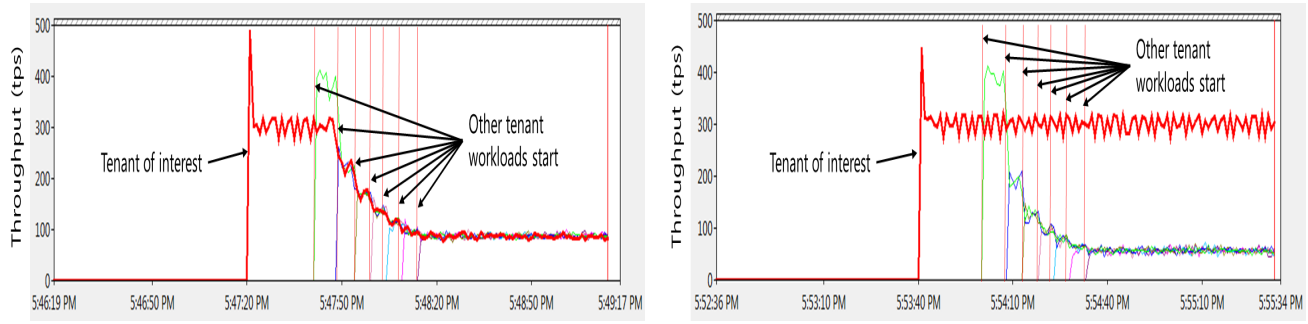
The goal of this demonstration is to illustrate when a tenant is running within a SQLVM, how its performance remains unaffected by the workloads of other co-located tenants. This demonstration will have three parts. In the first part, we will briefly outline the basic model of SQLVM and its resource reservations. This will provide the audience with the necessary background. In the second part, we will have a few scripted scenarios to demonstrate resource and performance isolation for a tenant executing within a SQLVM while other co-located tenants contend for resources. The final part will allow the audience to interact with the different SQLVM configurations and experience its impact on the tenant's performance.

3.1 Demo Setup

SQLVM is an abstraction implemented in Microsoft's SQL Azure. The demonstration will comprise an adapted version of SQL Azure running on our clusters which will act as the back-end. The front-end will emulate different tenant workloads executing against the database back-end. We will use a visualization tool (*Performance Monitor* for Windows) to monitor and visualize the various performance counters that are of interest.

To emulate variety in the tenant workloads typically observed in a multi-tenant cloud platform, our workload suite will comprise multiple workload types. Tenants in our workload suite will be able to execute TPC-C (OLTP-like workload),[1] TPC-H (DSS-like workload),[2] Dell DVD Store (e-commerce scenario),[3] and a custom workload for generating bursts of CPU- and I/O-intensive ac-

[1] `http://www.tpc.org/tpcc/`
[2] `http://www.tpc.org/tpch/`
[3] `http://linux.dell.com/dvdstore/`



(a) No performance isolation

(b) Tenant of interest running within a SQLVM

Figure 2: A screenshot of the demo experience and visualization. This figure illustrates the end-to-end throughput of the tenant of interested (shown in thick red line) when running without a SQLVM (sub figure (a)) and within a SQLVM (sub figure (b)).

tivity. A tenant's workload will be an instance of one of the workloads in the suite. We will have scripts to execute combinations of these workloads to emulate variety in tenant workloads typically observed in a multi-tenant platform.

3.2 Demo Script

The basic demo experience will focus on SQLVM's performance and resource isolation features. The basic demo setup will have multiple tenants co-located at a database server where each tenant can be individually configured to execute with or without a SQLVM. Tenants executing within a SQLVM has reservations for the critical resources within the database server while tenants without a SQLVM only have best-effort resource allocations. One of these tenants (T_1) will be our *tenant of interest*. We will monitor T_1's performance in terms of end-to-end throughput (transactions per second) and response times (or latency). T_1's workload starts executing first and then additional resource-intensive tenants are added into the system. We will consider two scenarios: first where T_1 is executing without a SQLVM, i.e., without any performance isolation, and the second where T_1 is requests performance isolation using SQLVM; in both scenarios, the remaining tenants execute without SQLVM. In the first scenario, the database server will provide best-effort resource allocation to all tenants and hence as other tenant workloads are added, T_1's performance will be impacted. On the other hand, when T_1 executes within a SQLVM, its resource allocation is isolated from other tenants, thus effectively isolating T_1's performance from the other tenants.

Figure 2 provides an illustration of the demo experience for the audience; this figure presents a screenshot of the visualization that will be used during the demo. Figure 2 shows the throughput (T_1's throughput is plotted in the thick red line) as time progresses and other tenants start executing their respective workloads. Figure 2(a) shows the throughput when T_1 is executing without a SQLVM while Figure 2(b) shows the throughput when T_1 is isolated within a SQLVM. During the demo, other similar visualization will monitor the response times and the critical resources such as the CPU utilization and I/O requests per second for each tenant.

The demonstration setup will also have additional scripted scenarios to illustrate other aspects of SQLVM. These scenarios include, but are not limited to: (i) providing low latency for bursty workloads; (ii) high consolidation ratios with a large number of tenants sharing a server and each executing within a SQLVM with resource reservations; (iii) low variance in performance even at high resource utilization levels when tenants are executing within a SQLVM; (iv) and resource allocation and end-to-end performance characteristics when multiple tenants execute within a SQLVM and at least one of the critical resources (such as CPU or I/O) at the server is overbooked; and (v) the effectiveness of metering in detecting violations when the resources are overbooked.

3.3 User Interactions with SQLVM

The amount of resources reserved in a SQLVM is dynamically configurable. For instance, if a tenant executing within a SQLVM requires additional CPU or I/O, the SQLVM configurations can be dynamically changed to provide the additional resources if they are available at the server. Similarly, all tenant workloads also have configuration parameters to vary the offered load and the number of concurrent database connections concurrently issuing workload for a tenant. In addition, a custom workload in the suite allows changing data access patterns and hence the resource usage of the workload. The demo setup will allow the audience to interact with the system to change the SQLVM or workload configurations and observe the corresponding impact on the tenant's performance.

Acknowledgments

Authors F. Li and H. Park contributed to SQLVM when visiting Microsoft Research. Badrish Chandramouli, Hamid Moussavi, and Vamsidhar Thummula have also contributed to the SQLVM project. Several members of the Microsoft SQL Azure group, including Peter Carlin, George Reynya, and Morgan Oslake, have provided valuable feedback that has influenced our work.

4. REFERENCES

[1] C. Curino, E. P. C. Jones, S. Madden, and H. Balakrishnan. Workload-aware database monitoring and consolidation. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 313–324, 2011.

[2] S. Krompass, A. Scholz, M.-C. Albutiu, H. A. Kuno, J. L. Wiener, U. Dayal, and A. Kemper. Quality of service-enabled management of database workloads. *IEEE Data Eng. Bull.*, 31(1):20–27, 2008.

[3] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan. Towards multi-tenant performance slos. In *Proc. 28th Int. Conf. on Data Engineering*, pages 702–713, 2012.

[4] V. Narasayya, S. Das, M. Syamala, B. Chandramouli, and S. Chaudhuri. SQLVM: Performance Isolation in Multi-Tenant Relational Database-as-a-Service. In *Proc. 6th Biennial Conf. on Innovative Data Systems Research*, 2013.

[5] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacigümüs. Intelligent management of virtualized resources for database systems in cloud environment. In *Proc. 27th Int. Conf. on Data Engineering*, pages 87–98, 2011.