Rackspace Cloud Databases and Container-based Virtualization

August 2012

J.R. Arredondo
@jrarredondo

INTRODUCTION

When Rackspace set out to build the Cloud Databases product, we asked many customers what they were looking for from a relational service in the cloud. The typical questions that customers asked back relate to performance and latency of the database: “How can you help me increase transactions per second?” or “How can you help me reduce my application latency?” We decided to make cost-effective performance the cornerstone of our engineering investments and set out to create a service that was purpose-built to get the most of out of the market’s more popular relational engines.

An important aspect of Cloud Databases implementation is its use of Container-based virtualization. This document is intended to give you a high level overview on our implementation decisions for the Rackspace Cloud Databases service, and its impact on IO, CPU, network and storage.

RELATIONAL DATABASES

Relational databases are one of the pillars of the computing stack. Organizations and businesses have depended on them for decades to store very important information for their operations, manage business transactions, maintain the integrity of their master data, and derive business intelligence. It is instructive to revisit they key aspects of a relational engine from a developer and operations point of view.

The relational model has stood the test of time as one of the most reliable pieces of technology that developers can depend on, and while other approaches become popular, the relational model is not going anywhere anytime soon. Developers and administrators have access to SQL as a standard language to interact with the database, allowing them to describe the terminology of their business in tables, relationships, restrictions, primary and secondary keys, and schemas; as well as to implement the business transactions that read or modify the state of those entities, using simple queries, enforce integrity rules through stored procedures and triggers, and perform complex aggregations through JOINs.

From an operational perspective, system administrators and database administrators can configure storage, memory, and networks to ensure an optimal operation for the database engine; and they can also execute important operational tasks such as monitoring the availability and performance of the engine, managing storage growth, backing up and restoring databases, or planning for replication and disaster recovery scenarios.

All of this resulted in databases becoming the foundation for highly dependable systems of record, data warehouses and other mission-critical applications for organizations. But in the process, database engines have become highly complex, and place high demands on hardware, system and network design in order to achieve the right quality of service, performance and availability that the users require.

RELATIONAL DATABASES AND TRADITIONAL VIRTUALIZATION

Due to the nature the job of a relational database engine, its performance is tightly dependent on unrestricted access to the hardware.

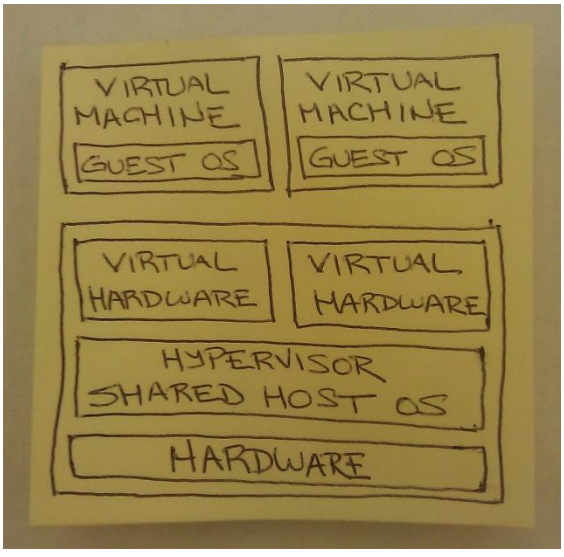
These are a few examples of how a database engine depends on key hardware components for its performance:

- **MEMORY** is used to cache tables, temporary datasets, internal binary tree representations of the indexes, and query execution plans; and is also required to process the background tasks and capture the background data required by administrators and developers to have visibility into the internal behavior and performance of a database.
- **STORAGE,** both local to the host or on the network, is used to maintain the logical and physical structured of the database, including tables, indexes, logs, the temporary data sets that are created with the execution of JOINs or SORT operations, long running transactions that may or may not be committed, and configuration information, among other.
- **CPU** power is critical to maintain threads and execute the triggers and stored procedures that perform the business rules and enforce the constraints specified by the developer or DBA, execute complex expression evaluations, parallelize queries, and to run the query optimization algorithms performed by the engine.
- **NETWORK** and **BANDWIDTH** are required for all distributed scenarios, read and write potentially large amounts of data from queries and JOINs, and for the client-server interactions between the application (the client) and the database engine and its storage.

As applications have increasingly moved to the cloud, virtualization has become a common approach to increase density, simplify manageability and provide cost-effective services. There have historically been two major approaches to virtualization, and both have limitations when it comes to hosting databases.

Hardware virtualization

Hardware virtualization works by using a virtual machine manager, commonly referred to as the hypervisor, to allow multiple guest operating systems to run on a single host computer. The key benefit of the hypervisor in this architecture is to allow the single host to be used by multiple virtual machines, in effect reducing the historical costs and waste associated when a single operating system would utilize only a small proportion of the host on which it was installed. There were other benefits associated with virtualization in general: organizations saw a reduction in the number of physical devices to purchase and more importantly, to manage; and they saw higher flexibility to deliver business services and applications because of the resulting device



independence, ability to run many different operating systems, the resulting consolidation of hardware, and new options to deliver more complex cluster architectures.

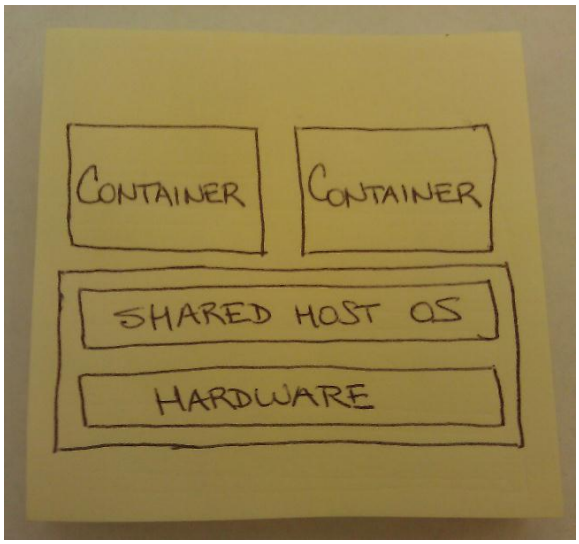
While this architecture had many benefits, its side effect was that it imposed performance penalties on the very same hardware components that databases require to operate properly, because a guest is effectively not aware that is being virtualized. For example, for a single write operation to disk that a guest OS needs to execute, typically a local driver would have to write that data to shared memory, and then through interrupt-driven interactions between guest (local) drivers and host OS drivers, the data would finally be written to a physical hard drive. Other common effects were saturation of network and network cards with multiple guests try to use them simultaneously. Recently, hypervisors have improved on this limitation by adding what amounts to a "pass through" implementation that allows direct access from a guest to the hardware without hypervisor intervention, but other restrictions remain. For example, every guest must have its own complete operating system stack, consuming significantly more memory. In addition, the design created incentives for some guests to seek to consume more resources than other ones and create what is known as the "noisy neighbor" effect where that guest negatively affects the performance of all others because it takes over and controls a disproportionate amount of resources.

Paravirtualization

A second popular architecture for virtualization is called Paravirtualization. In this model, the hardware does not have to provide support for virtualization. Instead, the guest and host operating systems effectively reduce the surface area among which they have to collaborate to work together more efficiently without the overhead of a hypervisor. Interactions with the disk and with the network become more efficient. However, all of this comes at the cost of OS compatibility, as the guest would need to run a modified version of the operating system and its kernel, which eliminates the advantage of being able to run all operating systems designed for that hardware.

CONTAINER-BASED VIRTUALIZATION

When we started to design our Cloud Databases service, we experimented with a number of approaches. We eventually chose container-based virtualization as the direction for our implementation. At the time, we observed that containers were not as widely used, but during our investigations we concluded that this was mostly due to the extra effort required to run the services and applications inside each container.



Container-based virtualization (also called Operating System virtualization) does not virtualize the hardware, but instead, isolates each guest in its own "container." It uses container templates to control the provisioning of the operating system of a guest. One key aspect of container-based virtualization is how the modified core kernel isolates each container and manages the host's resources to ensure they are appropriately distributed for each and all guests. In this case, "appropriately" means that each container receives the right amount of resources that it has "rights to" based on the service that the customer has purchased. This would typically vary, but in the case of Rackspace, we seek to manage quality of service metrics based on what customers purchase and use, while giving each customer a solid level of "Fanatical Support" services for their various needs.

This is how container-based virtualization manages each one of the key aspects of the hardware that a database requires for proper performance:

- **MEMORY**: Container-based virtualization controls how much memory is allocated to each container on a guaranteed basis. Depending on the implementation, typically this number would control other configuration items such as memory allocated for TCP buffers, kernel memory, as well as the behavior of the system under out-of-memory situations.
- **STORAGE**: Container virtualization has mechanisms to control the IO priority that containers will receive when execution IO operations. Typically, these different priorities result in more or less time to use the IO channels to access local storage.
- **CPU**: Container-based virtualization provides for flexible allocation of CPU time and allocations of containers to CPUs, by defining the number of cores and processors that a container is allowed to utilize, the proportion of CPU time that a container will receive in relation to other containers based on certain measures of weight, or maybe even hard limits on how much processor time a container will be given, among other things.
- **NETWORK** and **BANDWIDTH**: Container virtualization allows for control of the number of TPC and non-TPC sockets, connections and therefore network clients that a container can serve simultaneously, and as a result, also affects the amount of memory allocated for network buffers.

More importantly for hosted cloud scenarios, network utilization rules and controls must be put in place and managed to ensure that different customers are given the appropriate amount of bandwidth based on the quality of service levels to which they are subscribed.

BENEFITS OF CONTAINER-BASED VIRTUALIZATION

As we stated above, Rackspace's goal was to provide a unique, high performance relational service on the cloud. While our internal architecture and design is undoubtedly a significant contributor in achieving this stated goal, the flexibility and control of container-based virtualization allowed Rackspace to provide different levels of performance at different price points, drastically reduce the waste in compute power typically associated with traditional hardware virtualization or paravirtualization, and reduce the potential for the "noisy neighbor" effect.

More importantly, this architecture also allows customers to prioritize their investments, giving vital applications more resources, while controlling the costs of less important workloads. Customers that use our Cloud Sites product or who manage their own MySQL service on Cloud Servers have reported positive feedback to us when using Cloud Databases.

The overhead cost in resources used by container-based virtualization tends to be small, typically around 2%, whereas traditional hardware virtualization tends to utilize 10-30% of the available CPU resources. Obviously, these metrics are specific to our benchmarks and depend on the nature of the workload, the types of the queries executed, the network conditions during the tests, etc., so we advise against using them without context.

CLOSING

We will be publishing more specific benchmark data of our service as we have it available. For more information, please go to the Technical Resources section in our Cloud Databases page in our Rackspace.com website or directly at
http://www.rackspace.com/cloud/cloud_hosting_products/databases/