OPINION

Counting database costs: databases and elasticity

How database systems are finally lining up with the rest of the application stack to support modern elastic data centers, and deliver the associated cost benefits.

Max Pixel (CC0)

> *"Dictatorship and rigidity rarely work. Freedom and elasticity do."* –Robert Mondavi

Historically, enterprise databases and elasticity have been mutually exclusive: you could either maintain the trust and consistency you get from a database of record, or you could have the ability to adjust your database on demand in reaction to (or anticipation of) volume spikes, but you could not have both. However, in this era of digital transformation, today's business cannot afford to make a choice. Neither is dispensable and if the data center of the future is to deliver on its potential, then somehow the CIO needs to find a way to embrace both requirements.

ADVERTISING

In what follows, I aim to provide some insight on how database systems are finally lining up with the rest of the application stack to support modern elastic data centers, and deliver the associated cost benefits.

The elastic data center

In 2006 Amazon launched an offering that they had been building in Cape Town, South Africa, naming it "EC2" (Elastic Compute Cloud). They could have called it many things, perhaps "Virtual Compute Cloud" or "Resource Sharing Cloud" or any of a hundred other names. With significant foresight, they focused on it being "elastic". And in retrospect that was an inspired vision.

[Beware the 9 warning signs of bad IT architectureand see why these 10 old-school IT principles still rule. | Sign up for CIO newsletters.]

Amazon had figured out that clients of cloud services would ultimately want to think about services and not servers. Individual servers don't matter to the client, and nor do numbers of servers, sizes of servers, locations of servers, or any other implementation details. What matters is the combination of the service definition (API and related semantics), the quality of service (QoS/SLA), and the billing arrangements. Everything else is not only irrelevant, but is also free to change behind the curtain from moment to moment.

Amazon thought of this kind of system as "elastic", because it does not have finite limits. An elastic cloud "stretches" to meet demand. You can keep adding services, expanding services, adding users or inserting data. Of course, you can also reduce your usage of the service at any time. The client pays by-the-drink, and the costs will increase or decrease as the services are scaled up or down.

The key to the breakthrough is that the historical concept of sizing and provisioning servers is banished: No more under-utilized servers, no more overwhelmed services, no more waiting for the operations guys to deploy a new server for you, and no more resources wasted on tedious fork-lift upgrades. You only pay for the resources you use – you only tie-up the resources that you need, and those resources are continuously adjusted on-demand.

Besides easily increasing and decreasing capacity, an elastic data center also offers a radical reduction in planned and unplanned downtime. Elastic data centers are designed so that servers can fail and be replaced, or can be taken down and upgraded with minimal disruption. The new design pattern assumes failure, with automated, rapid and inexpensive recovery. This is significantly more robust, lower risk, and less expensive than trying to build servers that never fail.

[Looking to upgrade your career in tech? This comprehensive online course teaches you how.]

Amazon and other cloud systems have been wildly successful with the idea of elastic data centers, of course. Amazon reported nearly $10bn in AWS revenue in 2016 – not a bad achievement in 10 years.

The elastic application

Thousands of traditional applications have been moved to elastic clouds on a lift-and-shift basis, or through application modernization initiatives. And of course, cloud-native applications have been written specifically to take advantage of the new elastic infrastructure. We now assume that all applications of every description will run on a cloud sooner or later – the economics and the competitive advantages of elasticity are too compelling to ignore.

As clouds have matured, application architectures have evolved. They have become container and microservices based and managed within container ecosystems (eg Kubernetes). In many cases, applications or parts of applications have become "serverless." The further we go down this path, the greater the benefits we have seen in terms of operational costs/risks, application development velocity, and business agility. In many cases these benefits have translated directly into business opportunity, competitive advantage, and cost reduction.

As is common with elastic applications, Google's Gmail outperformed traditional competition. Gmail now has over 1.2bn active monthly users. Applications like Gmail (or Salesforce CRM, Amazon E-commerce, etc.) are elastic. They do not run on a single giant server behind the scenes. Instead they exploit elastic clouds to scale-out and scale-in.

If you start up Gmail and all the Gmail servers are busy, then a new server may be started to handle yourself and any additional users. In practice, user sessions are constantly being added and deleted, and the system is continually rebalancing connections as well as adding/deleting real and virtual nodes. How many servers does the 1.2bn user Gmail application use? Only as many as are needed at any given moment. It's elastic – that's the point!

As with elastic data centers, these applications inherit uptime benefits from their elasticity. If the server running your Gmail client went down, you would most likely not notice. Elasticity allows redundancy, quick restart, and rebalancing of load. Clients of the server that went down would simply be redirected to other (new or existing) servers.

Next-generation applications such as Netflix and Yelp are exploiting elasticity even further by adoptingmicroservices architectures, in which applications are decomposed into independent and loosely coupled services, all of which can be scaled up and down on-demand. Elasticity at the monolithic application level is a big win, but these much more modular application designs exploit elasticity at a finer-grained level. This approach has the potential to amplify the benefits of elastic applications discussed above (operational costs/risks, application development velocity, and business agility).

The inelastic database

The last 10 years have seen the rise of the elastic data center, and more recently the elastic application. But we do not yet have a fully elastic application stack. Most applications are still stuck with a huge challenge: Extremely inelastic database systems.

The traditional relational database (RDBMS) is a single-server system. They are certainly very powerful systems, driven by a succinct and expressive high-level language (SQL), protected by strong data guarantees (ACID transactions), and proven in a very wide range of application and management contexts. In a quite direct and concrete sense, we run the world on the RDBMS – Gartner estimates that RDBMSs represent 89% of the database market. But unfortunately, traditional RDBMS was designed in the 1980s and was targeted at single server deployments. It is a system designed for scale-up, not scale-out.

The traditional RDBMS is all about dedicated servers, fork-lift upgrades, and ugly workarounds for performance and scalability challenges. It is also about highly skilled administrators constantly analyzing, tweaking and optimizing the system and its workloads to keep your business up and running effectively.

As relates to uptime and reliability, we should note that the traditional RDBMS is a system in which single-points-of-failure are central to the design. It is not fundamentally designed to minimize planned and unplanned downtime, as elastic systems are. If you lose a server, a disk a disk controller, or a data center you should expect to lose data, transactions and/or uptime. If you need to upgrade something or even change a database schema you should expect to maintenance downtime. RDBMS vendors are certainly working on ways to get around these core limitations, but they are limitations that only exist because of the systems not being elastic in the first place.

No one could deny the three-decade run of success that the RDBMS has had in the client-server era, but it was achieved in the context of proprietary data centers with an expensive dedicated server for each application and each database. The traditional RDBMS is not designed for modern data centers or modern workloads, most particularly because it is not at all elastic.

The mainframe database was dominant prior to the client-server era, and the traditional RDBMS has been dominant prior to the elastic-cloud era. But both eras have been followed by workloads, data center designs, and cost models for which the previously dominant database designs were no longer well-suited.

The elastic database

We have a world that is moving inexorably and very rapidly towards elastic cloud architectures. On the other hand, we have database technologies that are single-server systems, very poorly suited to elastic cloud architectures. Clearly something must change – we need elastic databases.

There are adjustments and improvements being made by traditional database vendors, there are new approaches that challenge assumptions about the role of the database system, and there are even newer approaches targeted at addressing the core challenge of how to deliver classical relational database services in a fully elastic fashion.

In the coming weeks, I will continue to explore this topic, and provide an overview of the landscape as the industry grapples with the impedance mismatch between two of the most strategic elements of enterprise computing: Databases and Elasticity.

This article is published as part of the IDG Contributor Network. Want to Join?

Next read this: