AWS Database Blog

How Amazon DynamoDB adaptive capacity accommodates uneven data access patterns (or, why what you know about DynamoDB might be outdated)

by Richard Krog and Kai Zhao | on 13 AUG 2018 | in Amazon DynamoDB, Database | Permalink | Share

May 24, 2019, update: Amazon DynamoDB adaptive capacity is now instant, instead of having a 5–30 minute reaction time. Instant adaptive capacity is now on by default for all DynamoDB tables and global secondary indexes.

Amazon DynamoDB is a nonrelational database with high performance at any scale. It is a fully managed service that adapts to your throughput needs, with built-in security, backup, and data protection. More than 100,000 developers have chosen DynamoDB for mobile, web, gaming, ad tech, IoT, and other applications that need low-latency data access.

However, we still hear from customers who worry about their requests failing because of errors related to insufficient capacity, which they previously remedied by provisioning extra throughput. These customers are usually under the impression that DynamoDB is finicky about workloads where traffic isn't uniform across the primary key (also known as *hash key* or *partition key*), such as query patterns that read and write to some keys more than others.

In this post, we explain why capacity and provisioning in DynamoDB are no longer concerns. To do this, we first cover the basics of how DynamoDB shards your data across partitions and servers. Then, we highlight a feature called adaptive capacity that corrects the nonuniform workload issues that you might have experienced in the past. Last, to show adaptive capacity in action, we walk through an example application that you can run in your own AWS account.

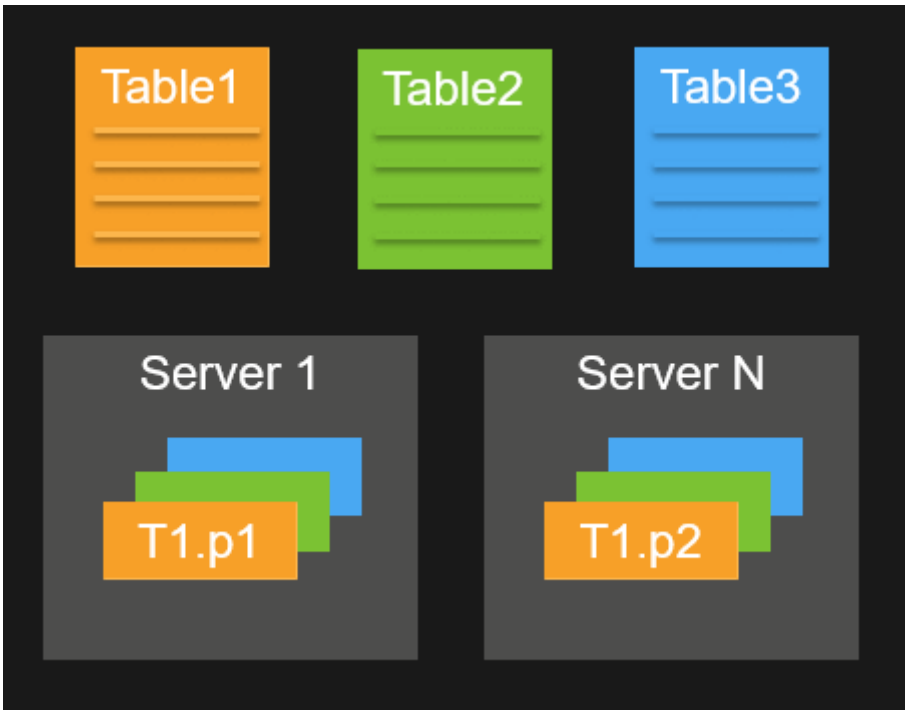
The DynamoDB approach to scaling

Note: If you are already familiar with DynamoDB partitioning and just want to learn about adaptive capacity, you can skip ahead to the next section.

you can skip ahead to the next section.

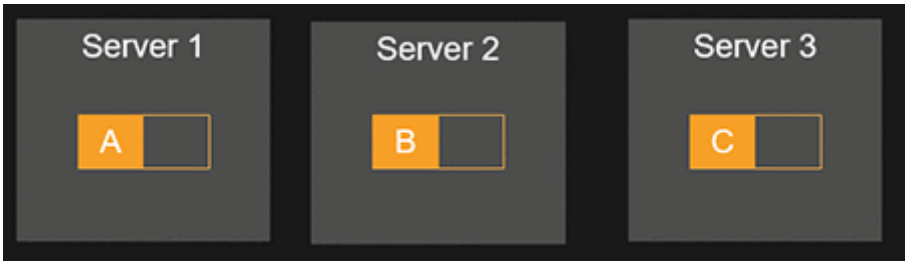
Let's start by understanding how DynamoDB manages your data. Like other nonrelational databases, DynamoDB horizontally shards tables into one or more partitions across multiple servers. But what differentiates using DynamoDB from hosting your own NoSQL database? Just as Amazon EC2 virtualizes server hardware to create a multitenant experience with the benefits of scale, efficiency, and lower cost, DynamoDB does the same with database hardware.

DynamoDB shards table partitions across physical servers transparently, as shown in the following example diagram. Table1 is sharded across two partitions (**T1.p1** and **T1.p2**), which are located on different servers.



To get started with DynamoDB, all you have to do is create a table and start sending reads and writes to it. You don't have to worry about selecting the right hardware (such as CPU, RAM, and storage) for your database node or cluster. DynamoDB handles the hardware resources behind the scenes. DynamoDB auto scaling automatically sets appropriate read and write throughput to serve your application's request rate. As your workload evolves, DynamoDB automatically reshards and dynamically redistributes your partitions in response to changes in read throughput, write throughput, and storage.

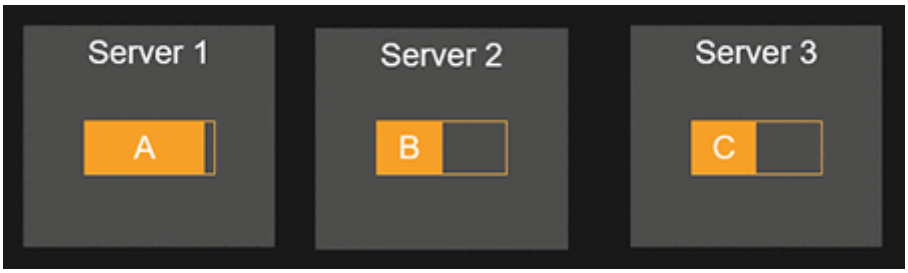
Let's look at an example of how DynamoDB resharding works, in this case, triggered by storage growth. Assume that you have a single DynamoDB table sharded across Partitions A, B, and C. These partitions are stored on three separate physical servers (Server 1, Server 2, and Server 3), as shown in the following illustration.



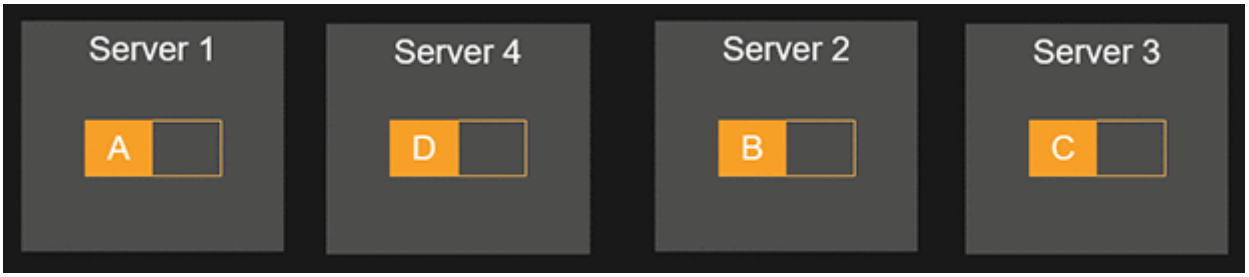
Note: DynamoDB actually would store this table's data across nine solid-state drives (not three). This is because

data is automatically replicated across three facilities in an AWS Region to provide fault tolerance in the event of a server failure or Availability Zone disruption. For this example, however, we are omitting replication for simplicity's sake.

In this case, the application is doing most of its writes to Partition A, meaning that Partition A's storage is nearly full, as shown in the following illustration.



Without requiring any input from you, DynamoDB automatically partitions Partition A into two parts: Partition A, which stays on Server 1, and Partition D, which is placed on Server 4. This change is transparent to your application, and DynamoDB automatically sends requests to the new partition.



Now that we've covered how partitions work, let's dive deeper into DynamoDB adaptive capacity.

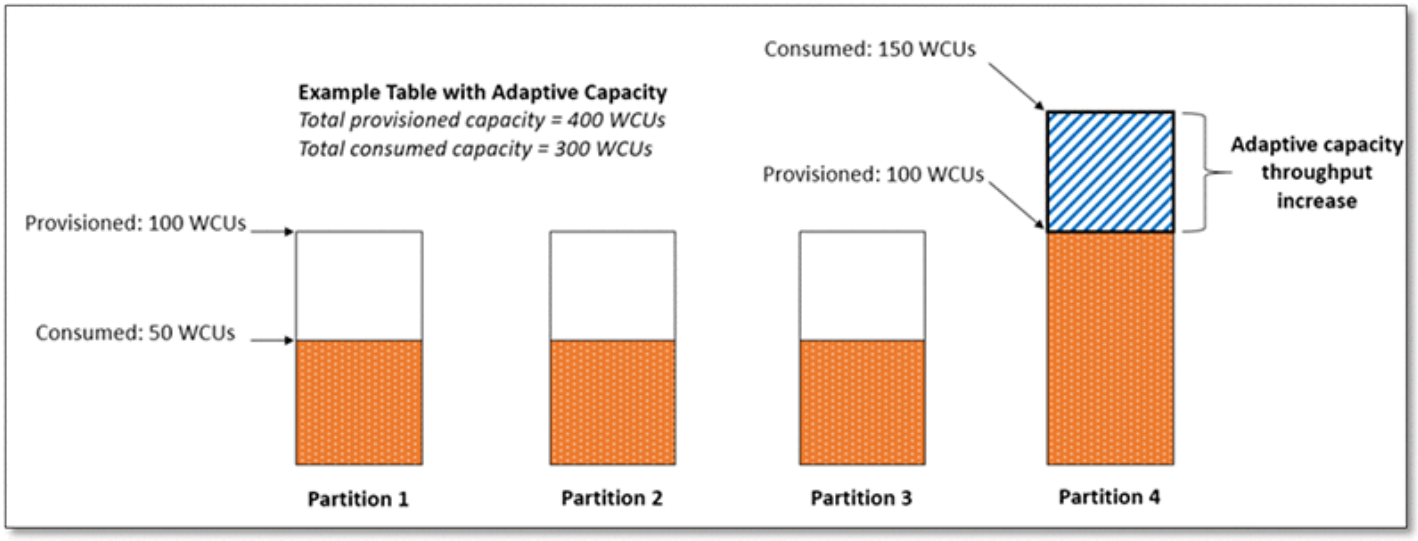
How adaptive capacity works

If you've used DynamoDB before, you're probably aware that DynamoDB recommends building your application to deliver evenly distributed traffic for optimal performance. That is, your requests should be evenly distributed across your primary key. This is because before adaptive capacity, DynamoDB allocated read and write throughput evenly across partitions. For example, if you had a table capable of 400 writes per second (in other words, 400 write capacity units, or "WCUs") distributed across four partitions, each partition would be allocated 100 WCUs. If you had a nonuniform workload with one partition receiving more than 100 writes per second (a hot partition), those requests might have returned a `ProvisionedThroughputExceededException` error.

In practice, it is difficult to achieve perfectly uniform access. To accommodate uneven data access patterns, DynamoDB adaptive capacity lets your application continue reading and writing to hot partitions without request failures (as long as you don't exceed your overall table-level throughput, of course). Adaptive capacity works by automatically increasing throughput capacity for partitions that receive more traffic. For a deep dive on adaptive capacity, see this AWS re:Invent 2017 breakout session video (63 minutes).

The following diagram shows an adaptive capacity example. This example table is provisioned with 400 WCUs that are evenly shared across four partitions, allowing each partition to sustain up to 100 writes per second. However, the application drives uneven traffic, with Partition 4 receiving 150 writes per second, even though

Partitions 1–3 receive only 50 writes per second. DynamoDB adaptive capacity automatically applies a "boost" to Partition 4, enabling it to consume more than its 100 WCU allocation. So the application continues to work normally and indefinitely despite uneven traffic.



Adaptive capacity is available by default for every DynamoDB table, so you don't need to explicitly enable or disable it. It is fully managed by DynamoDB, and you don't have to monitor any new Amazon CloudWatch metrics. When DynamoDB activates adaptive capacity on a table, the table continues to serve imbalanced traffic indefinitely, even if the workload remains unbalanced.

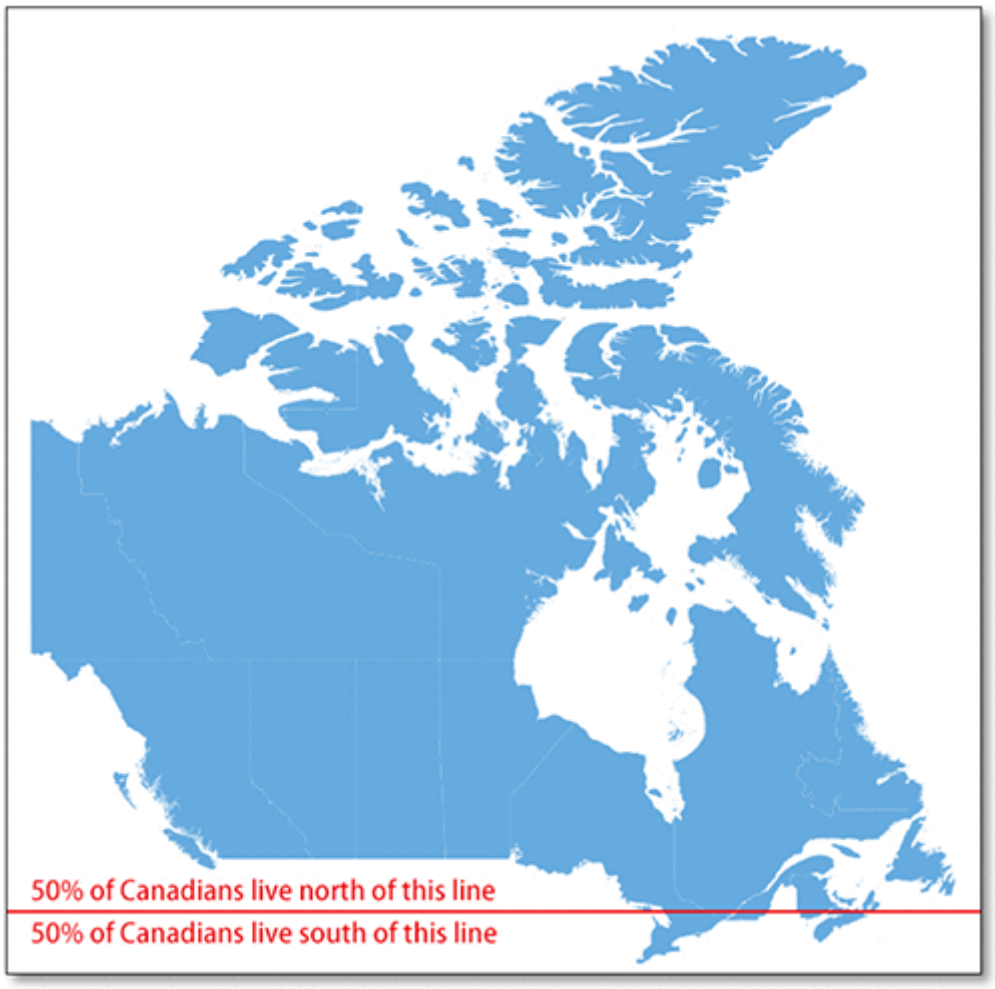
A Canadian census application – adaptive capacity in action

Let's explore how adaptive capacity responds to a typical application driving a nonuniform workload and eliminates `ProvisionedThroughputExceededException` errors caused by hot partitions. In this section, we describe the results of an example application that you can download and run yourself.

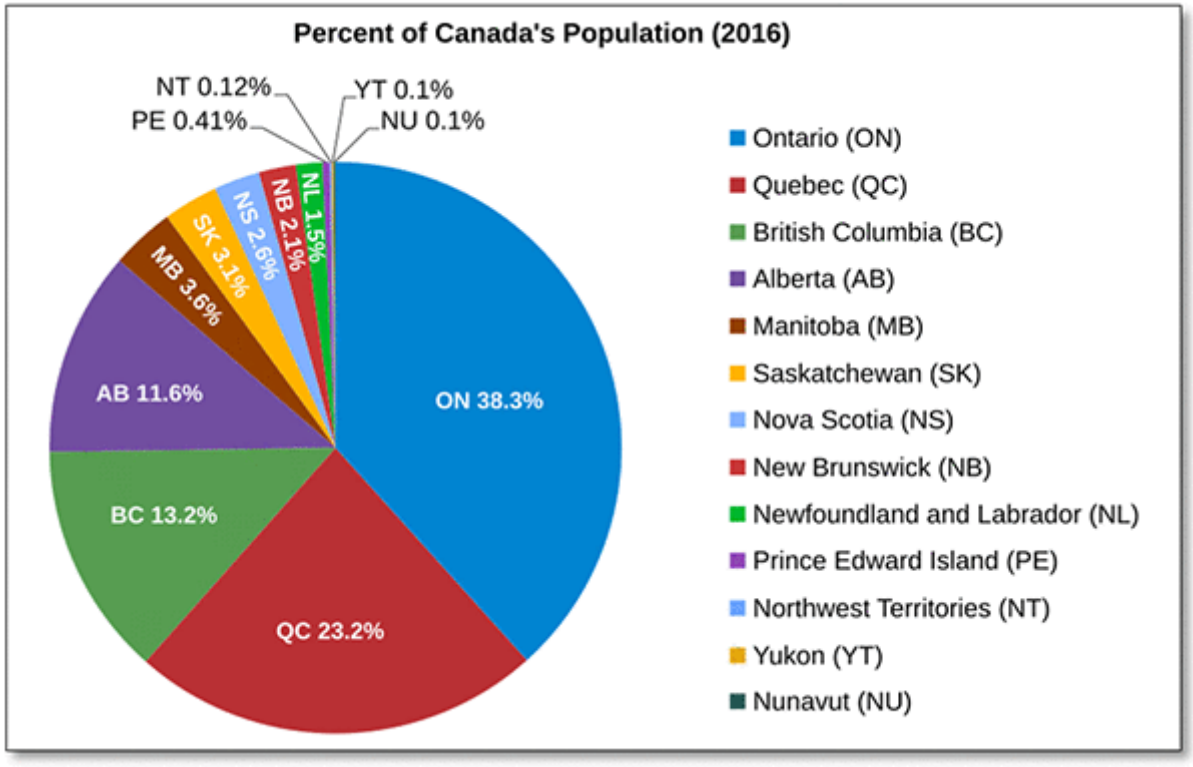
Scenario – The Canadian census

Let's assume that we're building an online census application for the Canadian population, a country that is spread across ten provinces and three territories. We choose to use DynamoDB and store the application's user responses in a table with the key schema shown in the following image (`Province` is the partition key and `ResponseId` is the sort key).

Now, let's assume that we're not particularly knowledgeable about Canada. Specifically, we didn't know the bit of trivia about Canada's population distribution that is shown in the following image.



Unfortunately, our partition key and sort key represent a poor choice of schema because the population is not uniformly distributed among provinces. This means that our DynamoDB access pattern will have uneven distribution of traffic because the more populous provinces will be written to more often. The following pie chart of the Canadian population shows that more than 60 percent of Canadians live in just two provinces: Ontario and Quebec.



Overview of the example census application

The example application simulates a web application for the Canadian census. First, the application creates a DynamoDB table with 3,000 WCUs and 3,000 read capacity units (RCUs), with `Province` as the primary key and `ResponseId` as the sort key. Provisioning this much throughput capacity results in four partitions being created. Subsequently lowering write throughput to 100 WCUs results in each of the four partition having 25 WCUs. Then, the application simulates receiving census replies from end users according to Canada's actual population distribution, at a rate of 70 replies per second. Each census reply writes a new item to the DynamoDB table created by the example application. The application emits a row of data every 10 seconds to represent the number of successful writes per province.

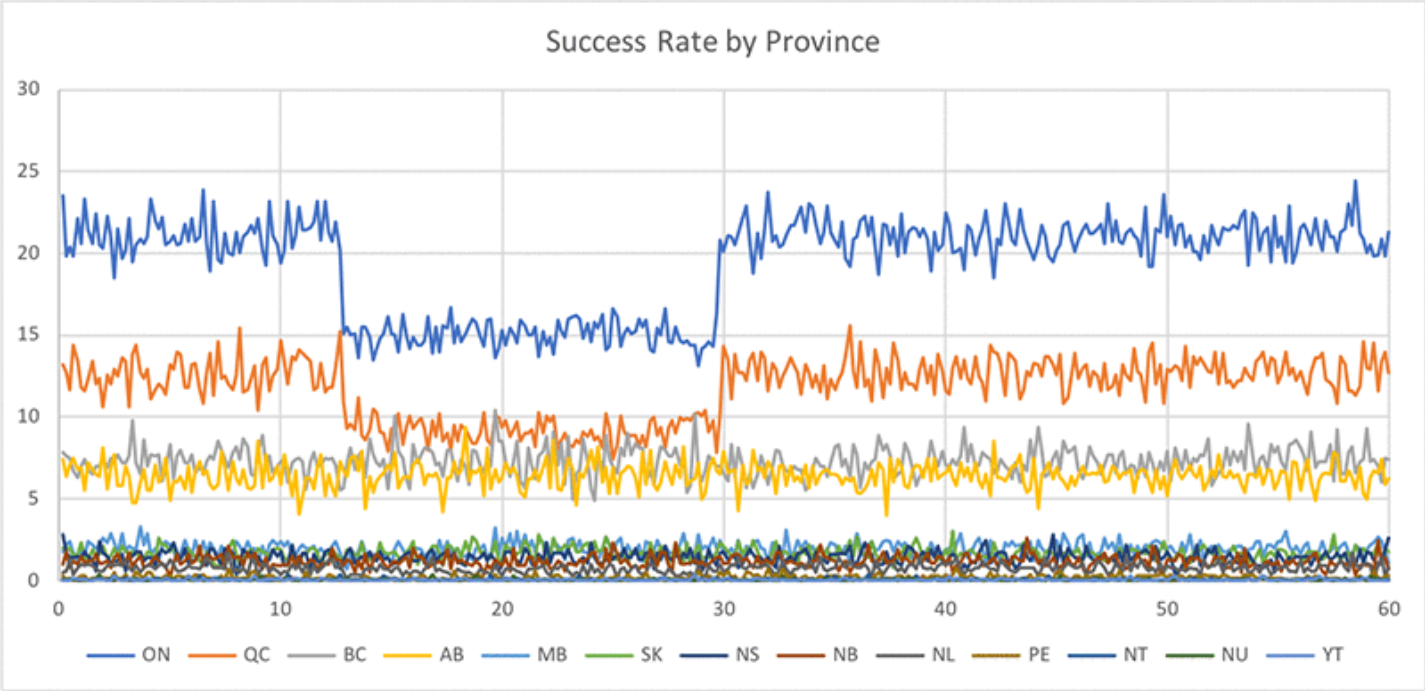
To run the example application yourself, go to this GitHub repository. Be aware of the following before running the application:

- To run the application, you need an AWS account and permissions to access DynamoDB.
- You might incur minor DynamoDB charges (around $10), depending on how long you run the simulation and whether you've already exhausted your monthly free-tier resources. The application requires 3,000 WCUs and 3,000 RCUs for about four hours.
- To clean up after the simulation completes, you will need to dial down or delete the DynamoDB table used by the application.

Running the example application and interpreting the results

Let's run the application to see how each province fares when we drive 70 writes per second randomly across them, proportional to the actual population distribution. The following graph shows a plot of the output. Note that the success rates for the blue `ON` line (Ontario) and the orange `QC` line (Quebec) experience a dip and

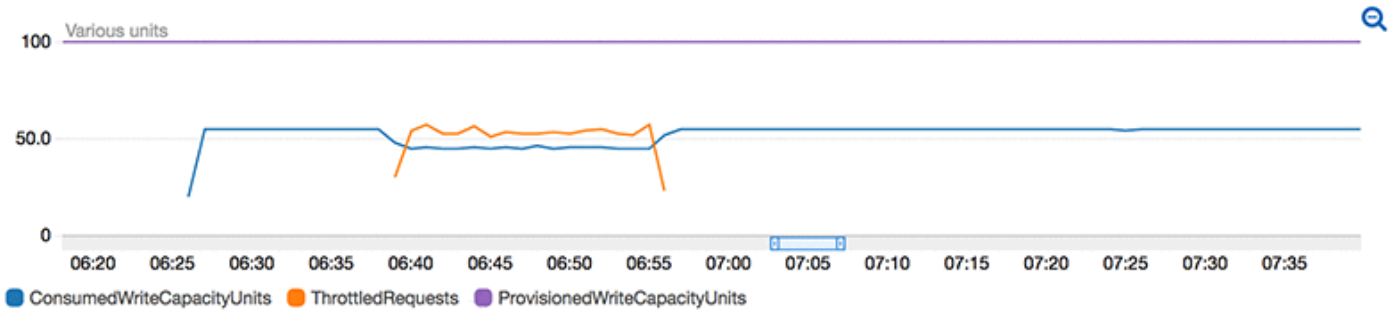
then return to normal.



Successful writes for Ontario and Quebec dropped after about 13 minutes because the string values for Ontario and Quebec mapped to the same partition by random chance. As a result, one partition was responsible for more than 60 percent of the table's traffic though only being provisioned with 25 percent of the throughput. The default 5-minute burst capacity helped (that's why the dip didn't happen right away) but was eventually exhausted because of the sustained traffic imbalance. Before adaptive capacity, the only remedy for these `ProvisionedThroughputExceededException` errors was to increase provisioned throughput or redesign the application for more uniform data access.

However, you also can see in the following chart that the successful writes recovered after about 30 minutes, which is when adaptive capacity kicked in. Without requiring any intervention, DynamoDB detected request failures that were triggered by insufficient partition-level throughput. Then, DynamoDB adjusted the table to better handle the imbalance. It typically takes 5–30 minutes from the time a table encounters request failures to when adaptive capacity restores normal performance.

For a zoomed-out view of what's happening, see the following CloudWatch graph. It represents successful write requests (the blue line) and throttled write requests (the orange line). Note that the same pattern plays out: The workload runs normally, throttling occurs due to insufficient partition-level activity, and then normal performance is restored by adaptive capacity.



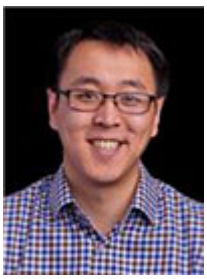
Wrapping up

We hope this blog post helps clarify how adaptive capacity enables DynamoDB to accommodate imbalanced workloads. With adaptive capacity, you don't need to overprovision read and write throughput. Combined with auto scaling, you can get the throughput you need when you need it, and dial down when your traffic decreases.

To learn more about DynamoDB adaptive capacity, see Understanding DynamoDB Adaptive Capacity and this AWS re:Invent 2017 breakout session video (63 minutes).

About the Authors

Richard Krog is a senior software developer at Amazon Web Services.



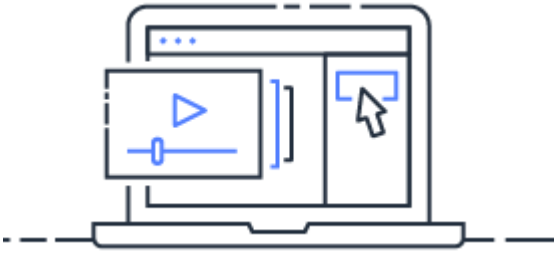
Kai Zhao is a senior product manager at Amazon Web Services.

TAGS: Amazon DynamoDB, DynamoDB

AWS Partner Network

Find an APN member to support your cloud business needs

Learn more »



AWS Training & Certifications

Free digital courses to help you develop your skills

Learn more »

Resources

Getting Started
What's New
Top Posts
Official AWS Podcast
AWS Case Studies

Follow

Twitter

Related Posts

Optimizing batch processing with custom checkpoints in AWS Lambda

Using AWS Lambda for streaming analytics

Detecting sensitive data in DynamoDB with Macie

BandLab welcomes users by the millions with AWS

ICYMI: Serverless pre:Invent 2020

Detect change points in your event data stream using Amazon Kinesis Data Streams, Amazon DynamoDB and AWS Lambda

Integrating Amazon ElastiCache with other AWS services: The serverless way

New – Export Amazon DynamoDB Table Data to Your Data Lake in Amazon S3, No Code Writing Required