Amazon DynamoDB – a Fast and Scalable NoSQL Database Service Designed for Internet Scale Applications

January 18, 2012 • 3119 words

Today is a very exciting day as we release Amazon DynamoDB, a fast, highly reliable and cost-effective NoSQL database service designed for internet scale applications. DynamoDB is the result of 15 years of learning in the areas of large scale non-relational databases and cloud services. Several years ago we published a paper on the details of Amazon's Dynamo technology, which was one of the first non-relational databases developed at Amazon. The original Dynamo design was based on a core set of strong distributed systems principles resulting in an ultra-scalable and highly reliable database system. Amazon DynamoDB, which is a new service, continues to build on these principles, and also builds on our years of experience with running non-relational databases and cloud services, such as Amazon SimpleDB and Amazon S3, at scale. It is very gratifying to see all of our learning and experience become available to our customers in the form of an easy-to-use managed service.

Amazon DynamoDB is a fully managed NoSQL database service that provides fast performance at any scale. Today's web-based applications often encounter database scaling challenges when faced with growth in users, traffic, and data. With Amazon DynamoDB, developers scaling cloud-based applications can start small with just the capacity they need and then increase the request capacity of a given table as their app grows in popularity. Their tables can also grow without limits as their users store increasing amounts of data. Behind the scenes, Amazon DynamoDB automatically spreads the data and traffic for a table over a sufficient number of servers to meet the request capacity specified by the customer. Amazon DynamoDB offers low, predictable latencies at any scale. Customers can typically achieve average service-side in the single-digit milliseconds. Amazon DynamoDB stores data on Solid State Drives (SSDs) and replicates it synchronously across multiple AWS Availability Zones in an AWS Region to provide built-in high availability and data durability.

History of NoSQL at Amazon – Dynamo

The Amazon.com ecommerce platform consists of hundreds of decoupled services developed and managed in a decentralized fashion. Each service encapsulates its own data and presents a hardened API for others to use. Most importantly, direct database access to the data from outside its respective service is not allowed. This architectural pattern was a response to the scaling challenges that had challenged Amazon.com through its first 5 years, when direct database access was one of the major bottlenecks in scaling and operating the business. While a service-oriented architecture addressed the problems of a centralized database architecture, each service was still using traditional data management systems. The growth of Amazon's business meant that many of these services needed more scalable database solutions.

In response, we began to develop a collection of storage and database technologies to address the demanding scalability and reliability requirements of the Amazon.com ecommerce platform. We had been pushing the scalability of commercially available technologies to their limits and finally reached a point where these third party technologies could no longer be used without significant risk. This was not our technology vendors' fault; Amazon’s scaling needs were beyond the specs for their technologies and we were using them in ways that most of their customers were not. A number of outages at the height of the 2004 holiday shopping season can be traced back to scaling commercial technologies beyond their boundaries.

Dynamo was born out of our need for a highly reliable, ultra-scalable key/value database. This non-relational, or NoSQL, database was targeted at use cases that were core to the Amazon ecommerce operation, such as the shopping cart and session service. Any downtime or performance degradation in these services has an immediate financial impact and their fault-tolerance and performance requirements for their data systems are very strict. These services also require the ability to scale infrastructure incrementally to accommodate growth in request rates or dataset sizes. Another important requirement for Dynamo was predictability. This is not just predictability of median performance and latency, but also at the end of the distribution (the 99.9th percentile), so we could provide acceptable performance for virtually every customer.

To achieve all of these goals, we needed to do groundbreaking work. After the successful launch of the first Dynamo system, we documented our experiences in a paper so others could benefit from them. Since then, several Dynamo clones have been built and the Dynamo paper has been the basis for several other types of distributed databases. This demonstrates that Amazon is not the only company than needs better tools to meet their database needs.

Lessons learned from Amazon’s Dynamo

Dynamo has been in use by a number of core services in the ecommerce platform, and their engineers have been very satisfied by its performance and incremental scalability. However, we never saw much adoption beyond these core services. This was remarkable because although Dynamo was originally built to serve the needs of the shopping cart, its design and implementation were much broader and based on input from many other service architects. As we spoke to many senior engineers and service owners, we saw a clear pattern start to emerge in their explanations of why they didn't adopt Dynamo more broadly: while Dynamo gave them a system that met their reliability, performance, and scalability needs, it did nothing to reduce the operational complexity of running large database systems. Since they were responsible for running their own Dynamo installations, they had to become experts on the various components running in multiple data centers. Also, they needed to make complex tradeoff decisions between consistency, performance, and reliability. This operational complexity was a barrier that kept them from adopting Dynamo.

During this period, several other systems appeared in the Amazon ecosystem that did meet their requirements for simplified operational complexity, notably Amazon S3 and Amazon SimpleDB. These were built as managed web services that eliminated the operational complexity of managing systems while still providing extremely high durability. Amazon engineers preferred to use these services instead of managing their own databases like Dynamo, even though Dynamo's functionality was better aligned with their applications' needs.

With Dynamo we had taken great care to build a system that met the requirements of our engineers. After evaluations, it was often obvious that Dynamo was ideal for many database use cases. But … we learned that engineers found the prospect of running a large software system daunting and instead looked for less ideal design alternatives that freed them from the burden of managing databases and allowed them to focus on their applications.

It became obvious that developers strongly preferred simplicity to fine-grained control as they voted "with their feet" and adopted cloud-based AWS solutions, like Amazon S3 and Amazon SimpleDB, over Dynamo. Dynamo might have been the best technology in the world at the time but it was still software you had to run yourself. And nobody wanted to learn how to do that if they didn't have to. Ultimately, developers wanted a service.

History of NoSQL at Amazon - SimpleDB

One of the cloud services Amazon developers preferred for their database needs was Amazon SimpleDB. In the 5 years that SimpleDB has been operational, we have learned a lot from its customers.

First and foremost, we have learned that a database service that takes away the operational headache of managing distributed systems is extremely powerful. Customers like SimpleDB's table interface and its flexible data model. Not having to update their schemas when their systems evolve makes life much easier. However, they most appreciate the fact that SimpleDB just works. It provides multi-data center replication, high availability, and offers rock-solid durability. And yet customers never need to worry about setting up, configuring, or patching their database.

Second, most database workloads do not require the complex query and transaction capabilities of a full-blown relational database. A database service that only presents a table interface with a restricted query set is a very important building block for many developers.

While SimpleDB has been successful and powers the applications of many customers, it has some limitations that customers have consistently asked us to address.

Domain scaling limitations. SimpleDB requires customers to manage their datasets in containers called Domains, which have a finite capacity in terms of storage (10 GB) and request throughput. Although many customers worked around SimpleDB's scaling limitations by partitioning their workloads over many Domains, this side of SimpleDB is certainly not simple. It also fails to meet the requirement of incremental scalability, something that is critical to many customers looking to adopt a NoSQL solution.

Predictability of Performance. SimpleDB, in keeping with its goal to be simple, indexes all attributes for each item stored in a domain. While this simplifies the customer experience on schema design and provides query flexibility, it has a negative impact on the predictability of performance. For example, every database write needs to update not just the basic record, but also all attribute indices (regardless of whether the customer is using all the indices for querying). Similarly, since the Domain maintains a large number of indices, its working set does not always fit in memory. This impacts the predictability of a Domain's read latency, particularly as dataset sizes grow. Consistency. SimpleDB's original implementation had taken the "eventually consistent" approach to the extreme and presented customers with consistency windows that were up to a second in duration. This meant the system was not intuitive to use and developers used to a more traditional database solution had trouble adapting to it. The SimpleDB team eventually addressed this issue by enabling customers to specify whether a given read operation should be strongly or eventually consistent.

Pricing complexity. SimpleDB introduced a very fine-grained pricing dimension called "Machine Hours." Although most customers have eventually learned how to predict their costs, it was not really transparent or simple.

Introducing DynamoDB

As we thought about how to address the limitations of SimpleDB and provide 1) the most scalable NoSQL solution available and 2) predictable high performance, we realized our goals could not be met with the SimpleDB APIs. Some SimpleDB operations require that all data for a Domain is on a single server, which prevents us from providing the seamless scalability our customers are demanding. In addition, SimpleDB APIs assume all item attributes are automatically indexed, which limits performance.

We concluded that an ideal solution would combine the best parts of the original Dynamo design (incremental scalability, predictable high performance) with the best parts of SimpleDB (ease of administration of a cloud service, consistency, and a table-based data model that is richer than a pure key-value store). These architectural discussions culminated in Amazon DynamoDB, a new NoSQL service that we are excited to release today.

Amazon DynamoDB is based on the principles of Dynamo, a progenitor of NoSQL, and brings the power of the cloud to the NoSQL database world. It offers customers high-availability, reliability, and incremental scalability, with no limits on dataset size or request throughput for a given table. And it is fast – it runs on the latest in solid-state drive (SSD) technology and incorporates numerous other optimizations to deliver low latency at any scale.

Amazon DynamoDB is the result of everything we've learned from building large-scale, non-relational databases for Amazon.com and building highly scalable and reliable cloud computing services at AWS. Amazon DynamoDB is a NoSQL database service that offers the following benefits:

- **Managed**. DynamoDB frees developers from the headaches of provisioning hardware and software, setting up and configuring a distributed database cluster, and managing ongoing cluster operations. It handles all the complexities of scaling and partitions and re-partitions your data over more machine resources to meet your I/O performance requirements. It also automatically replicates your data across multiple Availability Zones (and automatically re-replicates in the case of disk or node failures) to meet stringent availability and durability requirements. From our experience of running Amazon.com, we know that manageability is a critical requirement. We have seen many job postings from companies using NoSQL products that are looking for NoSQL database engineers to help scale their installations. We know from our Amazon experiences that once these clusters start growing, managing them becomes the same nightmare that running large RDBMS installations was. Because Amazon DynamoDB is a managed service, you won’t need to hire experts to manage your NoSQL installation—your developers can do it themselves.

- **Scalable**. Amazon DynamoDB is designed to scale the resources dedicated to a table to hundreds or even thousands of servers spread over multiple Availability Zones to meet your storage and throughput requirements. There are no pre-defined limits to the amount of data each table can store. Developers can store and retrieve any amount of data and DynamoDB will spread the data across more servers as the amount of data stored in your table grows.

- **Fast**. Amazon DynamoDB provides high throughput at very low latency. It is also built on Solid State Drives to help optimize for high performance even at high scale. Moreover, by not indexing all attributes, the cost of read and write operations is low as write operations involve updating only the primary key index thereby reducing the latency of both read and write operations. An application running in EC2 will typically see average service-side latencies in the single-digit millisecond range for a 1KB object. Most importantly, DynamoDB latencies are predictable. Even as datasets grow, latencies remain stable due to the distributed nature of DynamoDB’s data placement and request routing algorithms.

- **Durable and Highly Available**. Amazon DynamoDB replicates its data over at least 3 different data centers so that the system can continue to operate and serve data even under complex failure scenarios.

- **Flexible**. Amazon DynamoDB is an extremely flexible system that does not force its users into a particular data model or a particular consistency model. DynamoDB tables do not have a fixed schema but instead allow each data item to have any number of attributes, including multi-valued attributes. Developers can optionally use stronger consistency models when accessing the database, trading off some performance and availability for a simpler model. They can also take advantage of the atomic increment/decrement functionality of DynamoDB for counters.

- **Low cost**. Amazon DynamoDB's pricing is simple and predictable: Storage is $1 per GB per month. Requests are priced based on how much capacity is reserved: $0.01 per hour for every 10 units of Write Capacity and $0.01 per hour for every 50 units of Read Capacity. A unit of Read (or Write) Capacity equals one read (or write) per second of capacity for items up to 1KB in size. If you use eventually consistent reads, you can achieve twice as many reads per second for a given amount of Read Capacity. Larger items will require additional throughput capacity.

In the current release, customers will have the choice of using two types of keys for primary index querying: Simple Hash Keys and Composite Hash Key / Range Keys:

Simple Hash Key gives DynamoDB the Distributed Hash Table abstraction. The key is hashed over the different partitions to optimize workload distribution. For more background on this please read the original Dynamo paper.

Composite Hash Key with Range Key allows the developer to create a primary key that is the composite of two attributes, a “hash attribute” and a “range attribute.” When querying against a composite key, the hash attribute needs to be uniquely matched but a range operation can be specified for the range attribute: e.g. all orders from Werner in the past 24 hours, all log entries from server 16 with clients IP addresses on subnet 192.168.1.0

Performance Predictability in DynamoDB

In addition to taking the best ideas of Dynamo and SimpleDB, we have added new functionality to provide even greater performance predictability.

Cloud-based systems have invented solutions to ensure fairness and present their customers with uniform performance, so that no burst load from any customer should adversely impact others. This is a great approach and makes for many happy customers, but often does not give a single customer the ability to ask for higher throughput if they need it.

As satisfied as engineers can be with the simplicity of cloud-based solutions, they would love to specify the request throughput they need and let the system reconfigure itself to meet their requirements. Without this ability, engineers often have to carefully manage caching systems to ensure they can achieve low-latency and predictable performance as their workloads scale. This introduces complexity that takes away some of the simplicity of using cloud-based solutions.

The number of applications that need this type of performance predictability is increasing: online gaming, social graphs applications, online advertising, and real-time analytics to name a few. AWS customers are building increasingly sophisticated applications that could benefit from a database that can give them fast, predictable performance that exactly matches their needs.

Amazon DynamoDB's answer to this problem is "Provisioned Throughput." Customers can now specify the request throughput capacity they require for a given table. Behind the scenes, DynamoDB will allocate sufficient resources to the table to predictably achieve this throughput with low-latency performance. Throughput reservations are elastic, so customers can increase or decrease the throughput capacity of a table on-demand using the AWS Management Console or the DynamoDB APIs. CloudWatch metrics enable customers to make informed decisions about the right amount of throughput to dedicate to a particular table. Customers using the service tell us that it enables them to achieve the appropriate amount of control over scaling and performance while maintaining simplicity. Rather than adding server infrastructure and re-partitioning their data, they simply change a value in the management console and DynamoDB takes care of the rest.

Summary

Amazon DynamoDB is designed to maintain predictably high performance and to be highly cost efficient for workloads of any scale, from the smallest to the largest internet-scale applications. You can get started with Amazon DynamoDB using a free tier that enables 40 million of requests per month free of charge. Additional request capacity is priced at cost-efficiently hourly rates as low as $.01 per hour for 10 units of Write Capacity or 50 strongly consistent units of Read Capacity (if you use eventually consistent reads you can get twice the throughput at the same cost, or the same read throughput at half the cost) Also, replicated solid state disk (SSD) storage is $1 per GB per month. Our low request pricing is designed to meet the needs of typical database workloads that perform large numbers of reads and writes against every GB of data stored.

To learn more about Amazon DynamoDB its functionality, APIs, use cases, and service pricing, please visit the detail page at aws.amazon.com/DynamoDB and also the Developer Guide. I am excited to see the years of experience with systems such as Amazon Dynamo result in an innovative database service that can be broadly used by all our customers.