Chair for Decentralized Information Systems and Data Management
TUM School of Computation, Information and Technology
Technical University of Munich

Cloud Information Systems

Exercise 13

27th January 2025

Prof. Dr. Viktor Leis, M.Sc. Till Steinert

Last Exercise

- Next Exercise will be a Q&A Session
- If you have questions regarding calculations, please send them in before the exercise
- Please send us your questions **until Wednesday EOD** (e-mail to jana.vatter@tum.de or till.steinert@tum.de)

DynamoDB

- Fully-managed, distributed NoSQL database
- Stores document data (e.g., JSON-like) as key-value pairs
- Design Goals:
  - Predictable Performance
  - High Scalability and Availability
  - Serverless / fully-managed by AWS

Amazon DynamoDB: A Scalable, Predictably Performant, and Fully Managed NoSQL Database Service

Mostafa Elhemali, Niall Gallagher, Nicholas Gordon, Joseph Idziorek, Richard Krog
Colin Lazier, Erben Mo, Akhilesh Mritunjai, Somu Perianayagam ,Tim Rath
Swami Sivasubramanian, James Christopher Sorenson III, Sroaj Sosothikul, Doug Terry, Akshat Vig
dynamodb-paper@amazon.com
Amazon Web Services

Abstract

Amazon DynamoDB is a NoSQL cloud database service that provides consistent performance at any scale. Hundreds of thousands of customers rely on DynamoDB for its fundamental properties: consistent performance, availability, durability, and a fully managed serverless experience. In 2021, during the 66-hour Amazon Prime Day shopping event, Amazon systems - including Alexa, the Amazon.com sites, and Amazon fulfillment centers, made trillions of API calls to DynamoDB, peaking at 89.2 million requests per second, while experiencing high availability with single-digit millisecond performance. Since the launch of DynamoDB in 2012, its design and implementation have evolved in response to our experiences operating it. The system has successfully dealt with issues related to fairness, traffic imbalance across partitions, monitoring, and automated system operations without impacting availability or performance. Reliability is essential, as even the slightest disruption can significantly impact customers. This paper presents our experience operating DynamoDB at a massive scale and how the architecture continues to evolve to meet the ever-increasing demands of customer workloads.

1 Introduction

Amazon DynamoDB is a NoSQL cloud database service that supports fast and predictable performance at any scale. DynamoDB is a foundational AWS service that serves hundreds of thousands of customers using a massive number of servers located in data centers around the world. DynamoDB powers multiple high-traffic Amazon properties and systems including Alexa, the Amazon.com sites, and all Amazon fulfillment centers. Moreover, many AWS services such as AWS Lambda, AWS Lake Formation, and Amazon SageMaker are built on DynamoDB, as well as hundreds of thousands of customer applications.

These applications and services have demanding operational requirements with respect to performance, reliability, durability, efficiency, and scale. The users of DynamoDB rely on its ability to serve requests with consistent low latency. For DynamoDB customers, consistent performance at any scale is often more important than median request service times because unexpectedly high latency requests can amplify through higher layers of applications that depend on DynamoDB and lead to a bad customer experience. The goal of the design of DynamoDB is to complete *all* requests with low single-digit millisecond latencies. In addition, the large and diverse set of customers who use DynamoDB rely on an ever-expanding feature set as shown in Figure 1. As DynamoDB has evolved over the last ten years, a key challenge has been adding features without impacting operational requirements. To benefit customers and application developers, DynamoDB uniquely integrates the following six fundamental system properties:

DynamoDB is a fully managed cloud service. Using the DynamoDB API, applications create tables and read and write data without regard for where those tables are stored or how they're managed. DynamoDB frees developers from the burden of patching software, managing hardware, configuring a distributed database cluster, and managing ongoing cluster operations. DynamoDB handles resource provisioning, automatically recovers from failures, encrypts data, manages software upgrades, performs backups, and accomplishes other tasks required of a fully-managed service.

DynamoDB employs a multi-tenant architecture. DynamoDB stores data from different customers on the same physical machines to ensure high utilization of resources, enabling us to pass the cost savings to our customers. Resource reservations, tight provisioning, and monitored usage provide isolation between the workloads of co-resident tables.

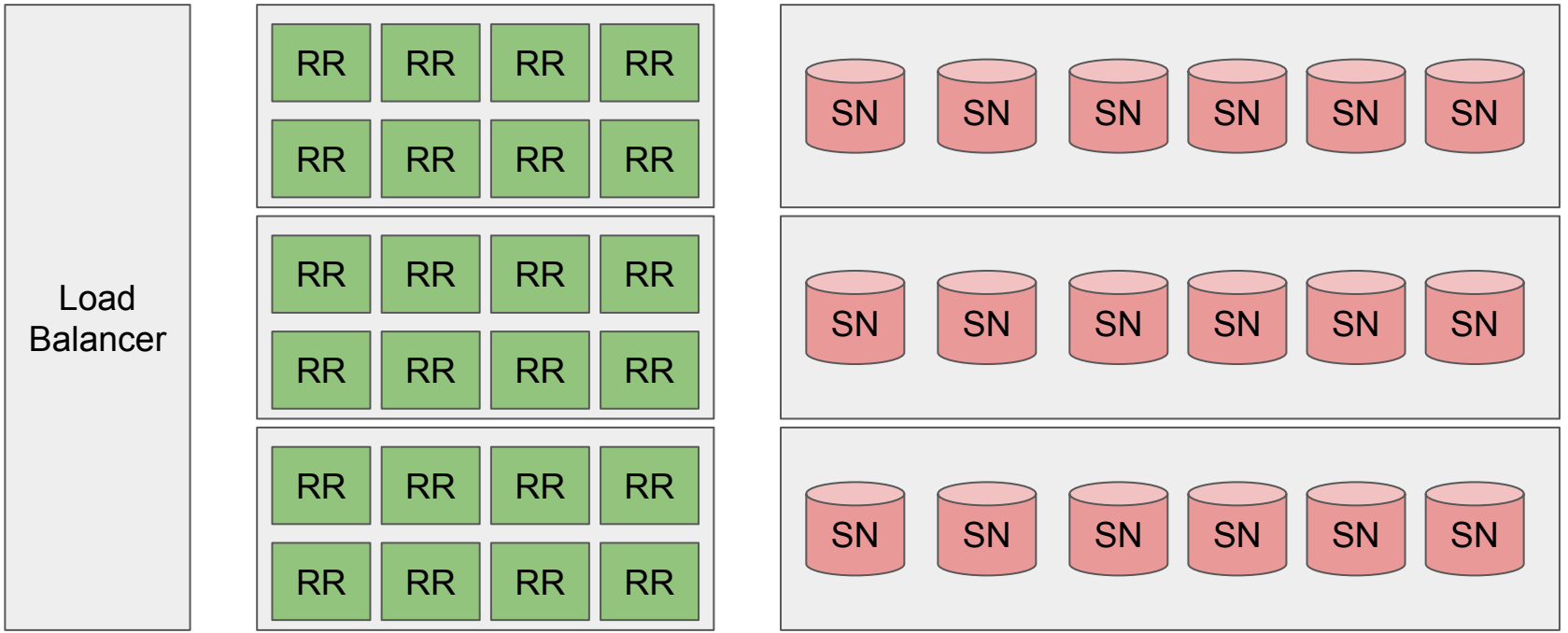
DynamoDB achieves boundless scale for tables. There are no predefined limits for the amount of data each table can store. Tables grow elastically to meet the demand of the customers' applications. DynamoDB is designed to scale the resources dedicated to a table from several servers to many thousands as needed. DynamoDB spreads an application's data across more servers as the amount of data storage and the demand for throughput requirements grow.

DynamoDB provides predictable performance. The simple

USENIX Association 2022 USENIX Annual Technical Conference 1037

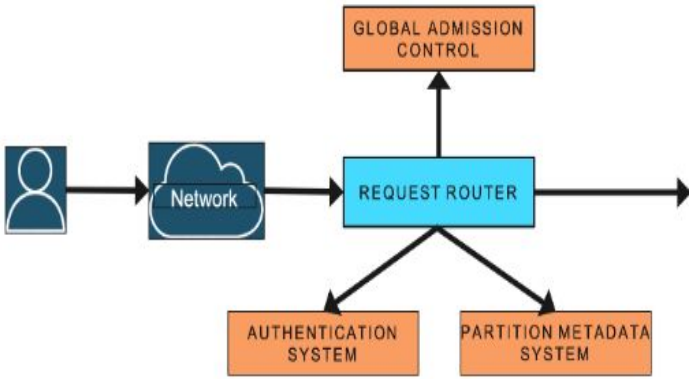
[Paper Link]

DynamoDB Architecture



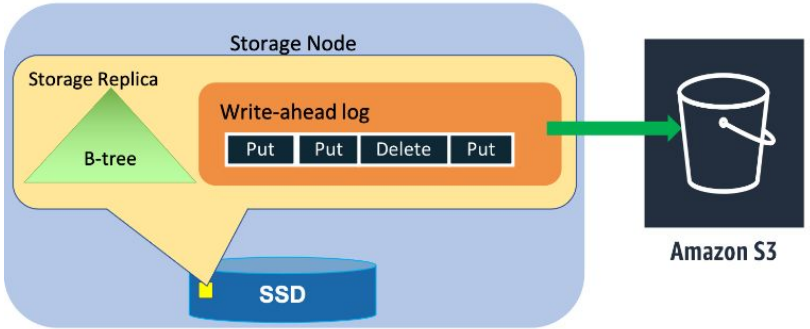
Request Router

- Through a Load Balancer, user requests are dispatched to a Request Router
- Three Services:
  - **Authentication System:** Ensure the request is valid (e.g., user is authorized to make request)
  - **Partition Metadata System:** Fetch routing information for request
  - **Global Admission Control:** Make sure the user has sufficient quota for the request (Token-Bucket algorithm)
- Request Routers are stateless -> Any RR can serve any API request

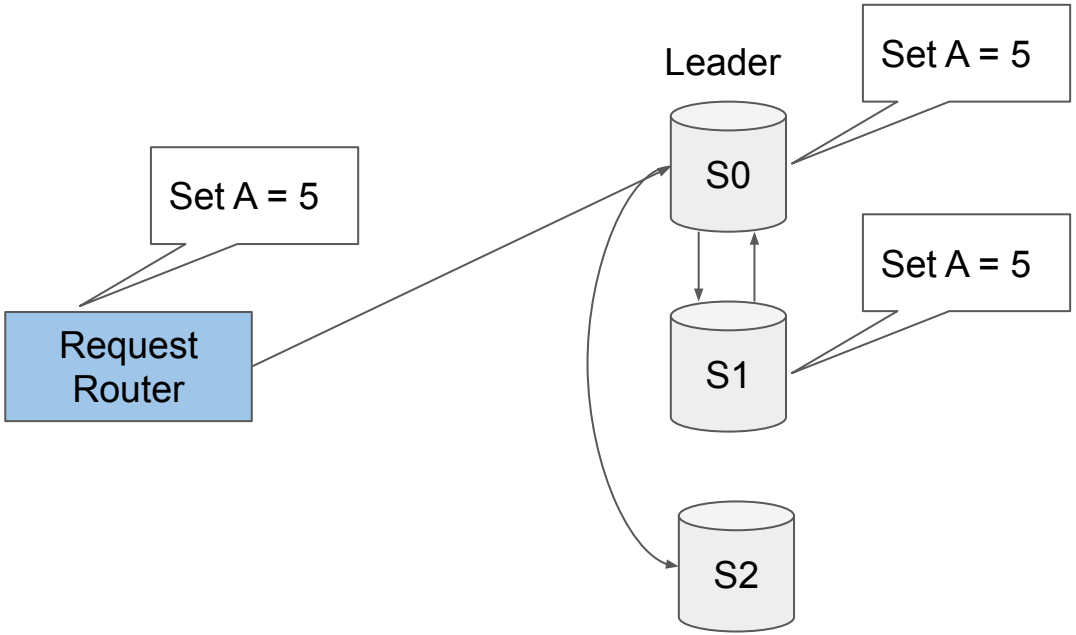


Storage Nodes

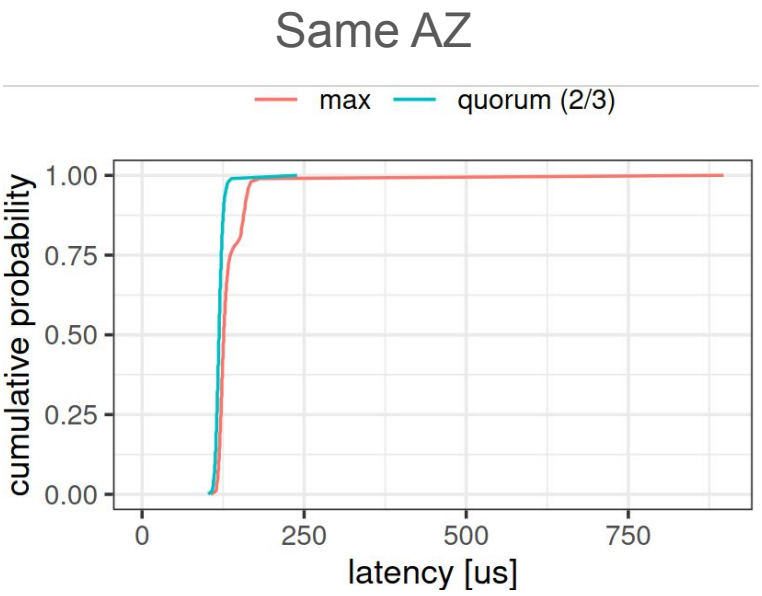
- Primary storage medium are SSDs (S3 is used for storing backups)
  - Based on SSD instances available at the time, DynamoDB most likely uses i3en instances
- Spread across multiple AZs
- Storage Nodes are multi-tenant and store partitions of different customers
- Partition Types
  - Leader
  - Replica
- Partition Group consists of one Leader and two Replicas
  - Leader sends periodic Heartbeat Messages
  - Missed HB message triggers Leader Election (Paxos)

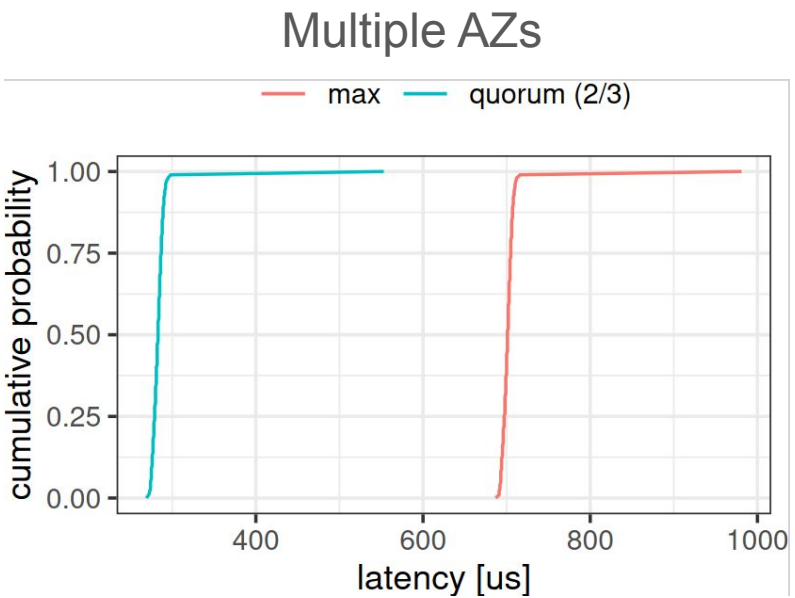


Write Requests

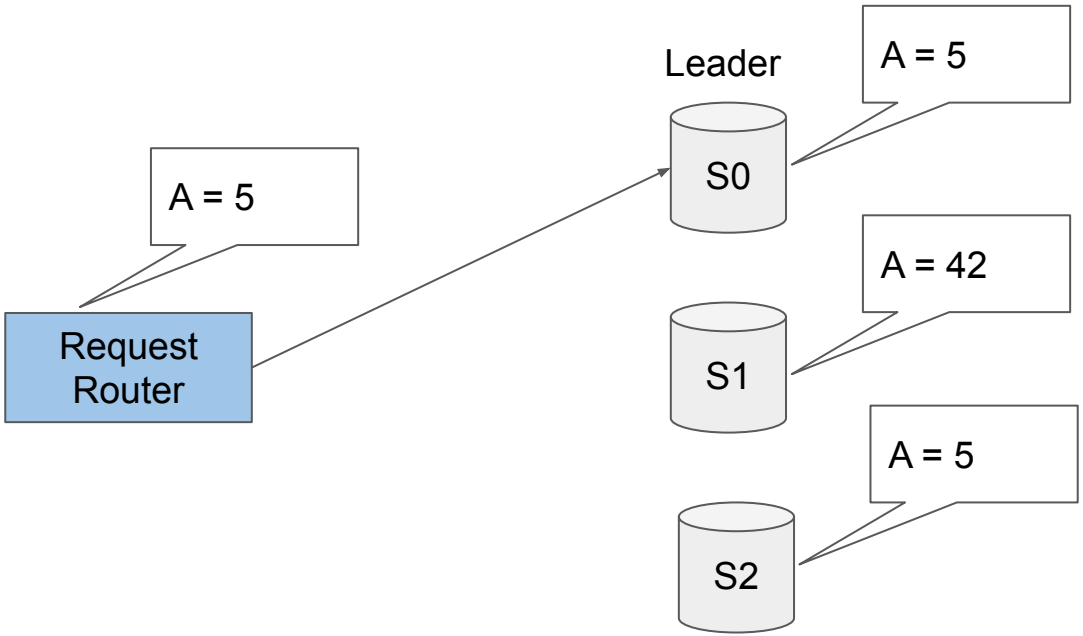


Reducing Network Tail Latency



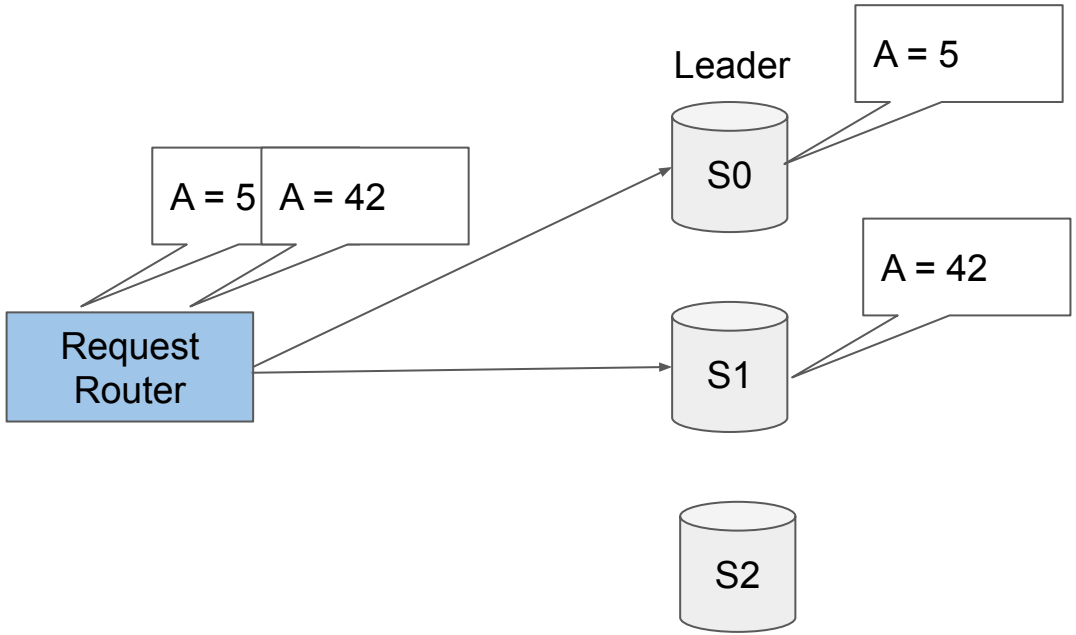


Strongly consistent Reads



By definition, the Leader has up-to-date state →Reading from Leader is sufficient

Eventually consistent Reads



- Depending on which Node we read, we may read **outdated** state
- There are no guarantees on *when* the other node will catch up

DynamoDB Pricing

Storage ($/TB/month)	$250
Writes [per 1KB]	1 WRU
Strongly-consistent Read [per 4KB]	1 RRU
Eventually-consistent Read [per 4KB]	0.5 RRU

- **Read Request Unit (RRU):** GetItem requests.
  - Charged in increments of 4KB
  - $0.125 per million read request units
- **Write Request Unit (WRU):** Any state changing API request (Insert, Update, Delete)
  - Charged in increments of 1KB
  - $0.625 per million write request units

3. Live Demo

4. Questions