Amazon DynamoDB: A Scalable, Predictably Performant, 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

Proceedings of the 2022 USENIX Annual Technical Conference
DynamoDB over the years
Key Aspects of DynamoDB

Predictability

Scalability

Availability

Consistency
Predictability
DynamoDB is a Key-Value Store

Operations: Get, Put, Update, Delete, …
Put

Network

REQUEST ROUTER

AUTHENTICATION SYSTEM

METADATA

STORAGE NODE

SSD

SSD

SSD

SSD

SSD

SSD

SSD

SSD

STORAGE NODE

SSD

SSD

SSD

SSD

SSD

SSD

SSD

STORAGE NODE

SSD

SSD

SSD

SSD

SSD

SSD

SSD

SSD

© 2022, Amazon Web Services, Inc. or its affiliates.
Issue: Multi-tenant Servers

100 Gets/sec to table A

500 Gets/sec To table B

© 2022, Amazon Web Services, Inc. or its affiliates.
Solution: Reserved Capacity

100 RCUs

500 RCUs

© 2022, Amazon Web Services, Inc. or its affiliates.
Token bucket algorithm

Refilled at RCU rate
100 tokens per second

Capacity = 100

Emptied 1 token per request*

*Tokens deducted depends on item size and consistency
Problem: Non-uniform request distribution over time

# of requests over time:
- Requested: < 100 RCUs and > 100 RCUs
- Provisioned: 100 RCUs

Throttling occurs when the number of requests exceeds the provisioned capacity.
Common solution: Over-provisioning

Requested
Provisioned

Waste!

500 RCUs

# of requests

Time
Bursting

Capacity = 300 * RCUs or 3000

Refilled at RCU rate
100 tokens per second
Bursting

# of requests

Bank

Spend

100 RCUs

Time

Requested

Provisioned
Scalability
Service at scale
<table>
<thead>
<tr>
<th>CustID</th>
<th>Customer information</th>
</tr>
</thead>
<tbody>
<tr>
<td>145783</td>
<td>{ name:“Bob”, city:”London”, …}</td>
</tr>
<tr>
<td>236294</td>
<td>{ name:“Sara”, city:”Tampa”, …}</td>
</tr>
<tr>
<td>333363</td>
<td>{ name:“Betty”, city:”Madison”, …}</td>
</tr>
<tr>
<td>445104</td>
<td>{ name:“James”, city:”Miami”, …}</td>
</tr>
<tr>
<td>523422</td>
<td>{ name:“Alex”, city:”London”, …}</td>
</tr>
<tr>
<td>643145</td>
<td>{ name:“Val”, city:”Seattle”, …}</td>
</tr>
<tr>
<td>723342</td>
<td>{ name:“Jeff”, city:”Toledo”, …}</td>
</tr>
</tbody>
</table>

Table
<table>
<thead>
<tr>
<th>Hash Value</th>
<th>CustID</th>
<th>Customer Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x9531</td>
<td>145783</td>
<td>{ name:“Bob”, city:”London”, …}</td>
</tr>
<tr>
<td>0x12A8</td>
<td>236294</td>
<td>{ name:“Sara”, city:”Tampa”, …}</td>
</tr>
<tr>
<td>0x6134</td>
<td>333363</td>
<td>{ name:“Betty”, city:”Madison”, …}</td>
</tr>
<tr>
<td>0x3391</td>
<td>445104</td>
<td>{ name:“James”, city:”Miami”, …}</td>
</tr>
<tr>
<td>0xF355</td>
<td>523422</td>
<td>{ name:“Alex”, city:”London”, …}</td>
</tr>
<tr>
<td>0xB082</td>
<td>643145</td>
<td>{ name:“Val”, city:”Seattle”, …}</td>
</tr>
<tr>
<td>0xEA8A</td>
<td>723342</td>
<td>{ name:“Jeff”, city:”Toledo”, …}</td>
</tr>
</tbody>
</table>
# Partitioning

<table>
<thead>
<tr>
<th>Hash Value</th>
<th>CustID</th>
<th>Customer Information</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x9531</td>
<td>145783</td>
<td><code>{ name:“Bob”, city:“London”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x12A8</td>
<td>236294</td>
<td><code>{ name:“Sara”, city:“Tampa”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3391</td>
<td>445104</td>
<td><code>{ name:“James”, city:“Miami”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6134</td>
<td>333363</td>
<td><code>{ name:“Betty”, city:“Madison”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xB082</td>
<td>643145</td>
<td><code>{ name:“Val”, city:“Seattle”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xEA8A</td>
<td>723342</td>
<td><code>{ name:“Jeff”, city:“Toledo”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xF355</td>
<td>523422</td>
<td><code>{ name:“Alex”, city:“London”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x12A8</td>
<td>236294</td>
<td><code>{ name:“Sara”, city:“Tampa”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3391</td>
<td>445104</td>
<td><code>{ name:“James”, city:“Miami”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6134</td>
<td>333363</td>
<td><code>{ name:“Betty”, city:“Madison”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xB082</td>
<td>643145</td>
<td><code>{ name:“Val”, city:“Seattle”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xEA8A</td>
<td>723342</td>
<td><code>{ name:“Jeff”, city:“Toledo”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xF355</td>
<td>523422</td>
<td><code>{ name:“Alex”, city:“London”</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Provisioning

300 read capacity units (RCU)

<table>
<thead>
<tr>
<th>Key</th>
<th>RCU</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12A8</td>
<td>236294</td>
<td><code>{ name:“Sara”, city:“Tampa”, ... }</code></td>
</tr>
<tr>
<td>0x3391</td>
<td>445104</td>
<td><code>{ name:“James”, city:“Miami”, ... }</code></td>
</tr>
<tr>
<td>0x6134</td>
<td>333363</td>
<td><code>{ name:“Betty”, city:“Madison”, ... }</code></td>
</tr>
<tr>
<td>0x9531</td>
<td>145783</td>
<td><code>{ name:“Bob”, city:“London”, ... }</code></td>
</tr>
<tr>
<td>0xB082</td>
<td>643145</td>
<td><code>{ name:“Val”, city:“Seattle”, ... }</code></td>
</tr>
<tr>
<td>0xEA8A</td>
<td>723342</td>
<td><code>{ name:“Jeff”, city:“Toledo”, ... }</code></td>
</tr>
<tr>
<td>0xF355</td>
<td>523422</td>
<td><code>{ name:“Alex”, city:“London”, ... }</code></td>
</tr>
</tbody>
</table>
# Provisioning

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12A8</td>
<td>236294</td>
<td>{ name:“Sara”, city:“Tampa”, …}</td>
</tr>
<tr>
<td>0x3391</td>
<td>445104</td>
<td>{ name:“James”, city:“Miami”, …}</td>
</tr>
<tr>
<td>0x6134</td>
<td>333363</td>
<td>{ name:“Betty”, city:“Madison”, …}</td>
</tr>
<tr>
<td>0x9531</td>
<td>145783</td>
<td>{ name:“Bob”, city:“London”, …}</td>
</tr>
<tr>
<td>0xB082</td>
<td>643145</td>
<td>{ name:“Val”, city:“Seattle”, …}</td>
</tr>
<tr>
<td>0xEA8A</td>
<td>723342</td>
<td>{ name:“Jeff”, city:“Toledo”, …}</td>
</tr>
<tr>
<td>0xF355</td>
<td>523422</td>
<td>{ name:“Alex”, city:“London”, …}</td>
</tr>
</tbody>
</table>

100 RCUs → 100 RCUs → 100 RCUs
**Problem: Non-uniform access across partitions**

<table>
<thead>
<tr>
<th>Partition</th>
<th>RCU Count</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12A8</td>
<td>250</td>
<td>{ name:“Sara”, city:”Tampa”, …}</td>
</tr>
<tr>
<td>0x3391</td>
<td>10</td>
<td>{ name:“James”, city:”Miami”, …}</td>
</tr>
<tr>
<td>0x6134</td>
<td>10</td>
<td>{ name:“Betty”, city:”Madison”, …}</td>
</tr>
<tr>
<td>0x9531</td>
<td>10</td>
<td>{ name:“Bob”, city:”London”, …}</td>
</tr>
<tr>
<td>0xB082</td>
<td>10</td>
<td>{ name:“Val”, city:”Seattle”, …}</td>
</tr>
<tr>
<td>0xEA8A</td>
<td>10</td>
<td>{ name:“Jeff”, city:”Toledo”, …}</td>
</tr>
<tr>
<td>0xF355</td>
<td>10</td>
<td>{ name:“Alex”, city:”London”, …}</td>
</tr>
</tbody>
</table>
Global admission control

300 RCUs

<table>
<thead>
<tr>
<th>Key (hex)</th>
<th>Value (dec)</th>
<th>Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12A8</td>
<td>236294</td>
<td>{ name:“Sara”, city:“Tampa”, …}</td>
</tr>
<tr>
<td>0x3391</td>
<td>445104</td>
<td>{ name:“James”, city:“Miami”, …}</td>
</tr>
<tr>
<td>0x6134</td>
<td>333363</td>
<td>{ name:“Betty”, city:“Madison”, …}</td>
</tr>
<tr>
<td>0x9531</td>
<td>145783</td>
<td>{ name:“Bob”, city:“London”, …}</td>
</tr>
<tr>
<td>0xB082</td>
<td>643145</td>
<td>{ name:“Val”, city:“Seattle”, …}</td>
</tr>
<tr>
<td>0xEA8A</td>
<td>723342</td>
<td>{ name:“Jeff”, city:“Toledo”, …}</td>
</tr>
<tr>
<td>0xF355</td>
<td>523422</td>
<td>{ name:“Alex”, city:“London”, …}</td>
</tr>
</tbody>
</table>
Global admission control

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12A8</td>
<td>236294</td>
<td>{ name: &quot;Sara&quot;, city: &quot;Tampa&quot;, … }</td>
</tr>
<tr>
<td>0x3391</td>
<td>445104</td>
<td>{ name: &quot;James&quot;, city: &quot;Miami&quot;, … }</td>
</tr>
<tr>
<td>0x6134</td>
<td>333363</td>
<td>{ name: &quot;Betty&quot;, city: &quot;Madison&quot;, … }</td>
</tr>
<tr>
<td>0x9531</td>
<td>145783</td>
<td>{ name: &quot;Bob&quot;, city: &quot;London&quot;, … }</td>
</tr>
<tr>
<td>0xB082</td>
<td>643145</td>
<td>{ name: &quot;Val&quot;, city: &quot;Seattle&quot;, … }</td>
</tr>
<tr>
<td>0xEA8A</td>
<td>723342</td>
<td>{ name: &quot;Jeff&quot;, city: &quot;Toledo&quot;, … }</td>
</tr>
<tr>
<td>0xF355</td>
<td>523422</td>
<td>{ name: &quot;Alex&quot;, city: &quot;London&quot;, … }</td>
</tr>
</tbody>
</table>
Global admission control

```
0x12A8  236294  { name:"Sara", city:"Tampa", ...}
0x3391  445104  { name:"James", city:"Miami", ...}
0x6134  333363  { name:"Betty", city:"Madison", ...}
0x9531  145783  { name:"Bob", city:"London", ...}
0xB082  643145  { name:"Val", city:"Seattle", ...}
0xEA8A  723342  { name:"Jeff", city:"Toledo", ...}
0xF355  523422  { name:"Alex", city:"London", ...}
```
Availability
Replication

- 0x12A8  236294  { name:“Sara”, city:“Tampa”, …}
- 0x3391  445104  { name:“James”, city:“Miami”, …}
- 0x6134  333363  { name:“Betty”, city:“Madison”, …}
- 0x9531  145783  { name:“Bob”, city:“London”, …}
- 0xB082  643145  { name:“Val”, city:“Seattle”, …}
- 0xEA8A  723342  { name:“Jeff”, city:“Toledo”, …}
- 0xF355  523422  { name:“Alex”, city:“London”, …}
### Partition Map

<table>
<thead>
<tr>
<th>Offset</th>
<th>Value</th>
<th>User Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000..0xFFFF</td>
<td>green1, green2, green3</td>
<td></td>
</tr>
<tr>
<td>0x7000..0xBFFF</td>
<td>orange1, orange2, orange3</td>
<td></td>
</tr>
<tr>
<td>0xC000..0xFFFF</td>
<td>pink1, pink2, pink3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Value</th>
<th>User Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12A8</td>
<td>236294</td>
<td>{ name:“Sara”, city:“Tampa”, …}</td>
</tr>
<tr>
<td>0x3391</td>
<td>445104</td>
<td>{ name:“James”, city:“Miami”, …}</td>
</tr>
<tr>
<td>0x6134</td>
<td>333363</td>
<td>{ name:“Betty”, city:“Madison”, …}</td>
</tr>
<tr>
<td>0x9531</td>
<td>145783</td>
<td>{ name:“Bob”, city:“London”, …}</td>
</tr>
<tr>
<td>0xB082</td>
<td>643145</td>
<td>{ name:“Val”, city:“Seattle”, …}</td>
</tr>
<tr>
<td>0xEA8A</td>
<td>723342</td>
<td>{ name:“Jeff”, city:“Toledo”, …}</td>
</tr>
<tr>
<td>0xF355</td>
<td>523422</td>
<td>{ name:“Alex”, city:“London”, …}</td>
</tr>
</tbody>
</table>
DynamoDB Global Tables

- Replicate table across regions
- Read and write anywhere
- Eventual convergence
- Last-writer-wins conflict resolution
Use Case: Disaster Recovery

Region A (Active)  
1. Put
2. Success

Region B (Standby)  
3. Put
4. Get
5. Item

© 2022, Amazon Web Services, Inc. or its affiliates.
Use Case: Disaster Recovery Failover

Region A (Active)  \rightarrow  Region B (Standby)

1. Put
2. Success
3. Put
4. Get
5. Item
6. Failover
7. Failover
8. Put
9. Success
10. Put

© 2022, Amazon Web Services, Inc. or its affiliates.
Use Case: Multi-Region Access

Region A (Oregon)

1. Put
2. Success

Region B (Ireland)

3. Get
4. Item
5. Put
6. Success

Region C (Singapore)

7. Put
8. Resolve Conflict!
9. Put
10. Put
11. Resolve Conflict!
12. Put
13. Resolve Conflict!
Intra-region vs. Cross-region Replication

![Diagram showing the comparison between Intra-region and Cross-region replication.]

**Intra-region Replication:**
- Strongly consistent
- Highly available
- Highly durable
- Partitioned
- Provisioned

**Cross-region Replication:**
- Concerns:
  - Write performance
  - Blast radius
  - Algorithm timeouts

© 2022, Amazon Web Services, Inc. or its affiliates.
Intra-region vs. Cross-region Replication

Intra-region:
- Strongly consistent
- Highly available
- Highly durable
- Partitioned
- Provisioned

Cross-region:
- Write in single region
- Eventually consistent
- Fault-tolerant
- Any number of regions
- Maintains DynamoDB properties
Consistency
Put

Network

REQUEST ROUTER

STORAGE NODE

LEADER STORAGE NODE

STORAGE NODE

© 2022, Amazon Web Services, Inc. or its affiliates.
Get

Strongly consistent

Eventually consistent

AMAZON DYNAMO DB: A SCALABLE, PREDICTABLY PERFORMANT, AND FULLY MANAGED NOSQL DATABASE SERVICE

© 2022, Amazon Web Services, Inc. or its affiliates.
Transactions

Facilitate the construction of correct and reliable applications that need to maintain multi-item invariants

Example: If Mary is Bob’s friend then Bob is Mary’s friend

Example: If Mary gives Bob $50, the total amount between them remains unchanged
Transaction Properties

Atomicity - execute all or nothing
Consistency - preserve correct state
Isolation - serialize concurrent operations
Durability - retain results permanently
DynamoDB Transactions

Execute sets of operations atomically and serializably for any items in any tables with predictable performance and no impact on non-transactional workloads.
Example: Money Transfer

person: Mary
balance: $25

person: Bob
balance: $100
Example: Money Transfer

person: Mary
balance: $75

person: Bob
balance: $50
Example: Money Transfer

mary-money = Get (person: “Mary”)
bob-money = Get (person: “Bob”)
Put (person: “Mary”, balance: mary-money + 50)
Example: Money Transfer

mary-money = Get (person: “Mary”)
bob-money = Get (person: “Bob”)

Put (person: “Mary”, balance: mary-money + 50)

Bob keeps his money
Example: Money Transfer

mary-money = Get (person: “Mary”)

bob-money = Get (person: “Bob”)

Put (person: “Mary”, balance: mary-money + 50)

Example: Money Transfer

mary-money = Get (person: “Mary”)

bob-money = Get (person: “Bob”)

bob-money = Get (person: “Bob”)
Put (person: “Bob”, bob-money + 100)

Put (person: “Mary”, balance: mary-money + 50)


Where’s my $100?
Standard Approach **Rejected**

**Explicit multi-step transactions**

```
TxBegin...
TxCommit
```

**Implicit singleton transactions**

```
TxBegin
Put (...)
TxCommit
```

**Multi-version Concurrency Control**

**Two-phase locking**

**Two-phase commit**

<table>
<thead>
<tr>
<th>Key</th>
<th>Timestamp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>400</td>
<td>&quot;current_value&quot;</td>
</tr>
<tr>
<td>A</td>
<td>322</td>
<td>&quot;old_value&quot;</td>
</tr>
<tr>
<td>A</td>
<td>50</td>
<td>&quot;original_value&quot;</td>
</tr>
<tr>
<td>B</td>
<td>100</td>
<td>&quot;value_of_b&quot;</td>
</tr>
</tbody>
</table>
DynamoDB Transactions

TransactGetItems (  
  Get (table: “T1”, key: k1),  
  Get (table: “T2”, key: k2),  
  Get (table: “T3”, key: k3)  
)

TransactWriteItems (  
  Put (table: “T1”, key: k1, value: v1),  
  Delete (table: “T2”, key: k2),  
  Update (table: “T3”, key: k3, value: +1),  
  Check (table: “T3”, key: k3, value: < 100)  
)
Shopping Example

- Customers
- Orders
- Inventory
Shopping Example

\textbf{TransactWriteItems (}

Check (table: "Customers", key: "Susie" EXISTS),
Check (table: "Inventory", key: "book-99", amount: \( \geq 5 \)),

\textbf{)}
### DynamoDB Transactions Architecture

1. **Request Router**
   - Receives TransactWriteItems from the **App**.
   - Sends TransactWriteItems to the **Transaction Coordinator**.
   - Notifies the **App** about Aborted/Completed transactions.

2. **Transaction Coordinator**
   - Coordinates transactions across nodes.
   - Notifies the **Request Router** and **Ledger** about Aborted/Completed transactions.

3. **Ledger**
   - Stores transaction outcomes.
   - Notifies the **Transaction Coordinator** about Aborted/Completed transactions.

4. **Nodes**
   - Process requests and updates.
   - Notify the **Transaction Coordinator** about the status of transactions.
Timestamp Ordering

Phil A. Bernstein, David W. Shipman, and James B. Rothnie, Concurrency Control in a System for Distributed Databases (SDD-1), *ACM TODS*, 1980.

Non-transactional Operations

- TransactWriteItems
  - Aborted/Completed
  - Put
  - Completed

- Ledger
  - 1. Prepare
  - 1. Accept/Reject
  - 2. Write
  - 2. Completed

- SN
  - 1. Prepare
  - 1. Accept/Reject
  - 2. Write
  - 2. Completed

- RR
  - Put
  - Completed

- TC
  - Aborted/Completed
  - Put
Take Away

DynamoDB evolved to meet customer needs while improving on its fundamental characteristics: predictability, scalability, availability, and consistency.
Thank you!