NoSQL in the Cloud with Windows Azure Table

Jai Haridas (Twitter: @jaiharidas)
Dev Manager
Windows Azure Storage - Microsoft
Objectives

• Introduction to Windows Azure Storage
• Windows Azure Table – Deeper dive
• Data Modeling for Scale
Why NoSQL?

• Large demand for economically accessing structured data at Internet scale
  o Distributed store
  o Auto scale
  o Flexible schema

• Relational databases
  o Scale up your servers: But gets expensive very soon
  o Manually Shard: But now you lose certain relational capabilities across shards
  o Maintenance requires DBA expertise
Windows Azure Storage –
A Quick Introduction
Windows Azure Storage Characteristics

• A “pay for what you use” cloud storage system
  o **Durable**: Store multiple replicas of your data
    • Local replication:
      – Synchronous replication before returning success
    • Geo replication:
      – Replicated to data center at least 400+ miles apart
      – Asynchronous replication after returning success to user.
  o **Available**: Multiple replicas provide fault tolerance
  o **Scalable**: Automatically partitions data across servers to meet traffic demands
    o **Strong consistency** by default
• 880K requests/s at peak & 4+ Trillion objects
• Great performance for low transaction costs!
Windows Azure Storage Characteristics

• Various Abstractions
  o **Blobs**: Store files and metadata associated with it
  o **Queues**: Durable asynchronous messaging system to decouple components
  o **Tables**: A strongly consistent NoSQL structured store that auto scales
  o **Drives and Disks**: Network mounted durable drives available for applications in the cloud

• All abstractions backed by same store
  o Same feature set across all abstractions (geo, durability, strong consistency, auto scale, monitoring, partitioning logic etc.)
  o Reduce costs by blending different characteristics of each abstraction

• Easy to use and open REST APIs
  o Client libraries in Java, Node.js, PHP, .NET etc.
Case Study: Bing Realtime facebook/twitter search ingestion engine

- Facebook/Twitter data stored into blobs
- Ingestion engine process blobs
  - Annotate with auth/spam/adult scores, content classification, expands links, etc
  - Uses Tables heavily for indexing
- Queues to manage work flow
- Results stored back into blobs
- Bing takes resulting blobs and folds into search index

**Bing Ingestion Engine (Azure Service)**

Index Facebook/Twitter data within 15 seconds of update

**Windows Azure Queues**

**Windows Azure Tables**

- peak 40,000 Requests/sec
- 2~3 billion Requests per day
- Took 1 dev 2 months to design, build and release to production
Deeper Dive into Windows Azure Tables
Windows Azure Table – Model

http://<account>.table.core.windows.net/

ACCOUNT

contoso

Table

Blogs

PartitionKey='uid1'
RowKey='Blog:002'
Rating='2'

Videos

PartitionKey='uid1'
RowKey='Comment:002:003'
Committed='1'

Entities

PartitionKey='uid2'
RowKey='pic.wmv'
Rating='1'
Windows Azure Table – Basics

• Authentication
  o Owner only access by default
  o Control access using pre-signed URLs

• Flexible schema
  o Tables – No schema associated
  o Store different entity types in same table

• Single Index
  o (AccountName, TableName, PartitionKey, RowKey) => Clustered Index
  o Defines sort order
  o PartitionKey – Mandatory property which defines partitioning
  o RowKey – Mandatory property which defines uniqueness within a given partition
Windows Azure Table – Basics

• Optimistic concurrency
  ○ Timestamp – A read only property maintained by server

• Partitioning
  ○ (AccountName, TableName, PartitionKey) => Partitioning key
  ○ Range based partitioning
  ○ Users can achieve hash based: PartitionKey = Hash(PartitionKey Value)

• Atomic Transactions
  ○ Entities with same partitioning key can be part of single batch request

• Exposed via RESTful APIs
  ○ OData protocol
  ○ Java, Node.js, PHP, .NET client libraries
## Windows Azure Table – Auto Scaling

<table>
<thead>
<tr>
<th>Account Name</th>
<th>Table Name</th>
<th>PartitionKey</th>
<th>RowKey</th>
</tr>
</thead>
<tbody>
<tr>
<td>contoso</td>
<td>Locations</td>
<td>Alaska</td>
<td>0001</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>contoso</td>
<td>Locations</td>
<td>New York</td>
<td>0901</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>contoso</td>
<td>Locations</td>
<td>Georgia</td>
<td>0001</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>contoso</td>
<td>Locations</td>
<td>New York</td>
<td>0901</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>contoso</td>
<td>Locations</td>
<td>Washington</td>
<td>0500</td>
</tr>
<tr>
<td>contoso</td>
<td>Locations</td>
<td>New York</td>
<td>0801</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>contoso</td>
<td>Locations</td>
<td>Washington</td>
<td>0500</td>
</tr>
</tbody>
</table>

### Partitioning
- Alaska - Washington
- Georgia - New Mexico
- Alaska - Florida
- Georgia - Washington
- New York - Washington

### Partition Servers
- Partition Server 1
- Partition Server 2
- Partition Server 3
Data Modeling for Scale
Data Modeling for Scale
Where do I Start?

• How large can partitions get?
  o Strive for PartitionKey that does not create hotspots
  o No size limit on partition
    ▪ Every entity can have different PartitionKey value
      – User table for popular movie streaming site may choose UserId
        » Avoids hotspots as user IDs are randomly distributed
    ▪ All entities can have same PartitionKey value
      – Genre table for movies may choose to have Genre enum listed with same PartitionKey value
        » Caching of rarely changing data can reduce hotspots
Data Modeling for Scale
Where do I Start?

- Start with following questions:
  - What objects do I need to deal with?
  - What are the significant queries that I need?
    - Determines candidates for “keys”
      - Retrieve list of movies released by week
        - PartitionKey == YYYY-MM-Week#
  - What are the atomic transactions that I need?
    - Atomic batch transactions allowed on entities having same (AccountName, TableName, PartitionKey) value
      - Transactional consistency
      - Batch to reduce round trips
      - Example: User gets rewards point with every checkout
        - Table maintains rewards info and currently checked out movies
        - Rewards info and rental info maintained in an atomic transaction
Data Modeling for Scale
Composite keys

• Single index provided by system but hierarchical pivots required
• Concatenate properties to get composite key
• Prefixes are more critical
  o Retrieve list of movies released recently by genre
    ▪ PartitionKey == Genre-YYYY-MM-Week#
    ▪ RowKey == MovieName
Data Modeling for Scale
Denormalization is key!

• Resultset pivoted by different keys
  o Retrieve list of movies released by genre
    ▪ PartitionKey == Genre & RowKey == MovieName
  o Retrieve list of movies with a given actor
    ▪ PartitionKey == Actor & RowKey == MovieName

• Store data twice – once for “by genre” lookup and once for “by actor”
  o Duplicate just required partial data
  o For larger entities, store pointer to fact table
    ▪ “By Genre” is the fact table that stores synopsis, thumbnail etc.
    ▪ “By Actor” just stores the properties like movie name, genre, rating etc. that are required to display results
      ▪ If user selects a movie – lookup for the movie “by genre” to retrieve synopsis, thumbnail etc.
Data Modeling for Scale
Index Table Pattern

• Partition level index
  o Entity group transaction to achieve consistent indexes
  o Example: List recent movies by Genre; List recent movies by rating
    ▪ Store 2 entities with PartitionKey == YYYY-MM-Week#
    ▪ One with RowKey == Genre-MovieName
    ▪ Other with RowKey == Rating-MovieName
Data Modeling for Scale
Index Table Pattern

• Indexes across Partitions
  o Eventual consistent indexes
    ▪ Add a message into queue that will add rows into index tables
    ▪ Add row into fact table
    ▪ Example: “By Actor” and “By Genre” queries.
      – Add message to queue which will update “By Actor” table if movie exists in fact table
      – Add movie to “By Genre” table
  o Usually Consistent indexes
    ▪ Add a message into queue that will fix consistency on failures
    ▪ Add row into fact table
    ▪ Add row into index tables
Data Modeling for Scale
What happened to my Joins?

• Application responsible for joins
  o User info table and rental info table
    ▪ When user logs in
      – Select * where PartitionKey = <UserId> from UserTable
      – Select * where PartitionKey = <UserId> from RentalTable
Data Modeling for Scale
What happened to my Joins?

• Shape data based on access pattern
  o Use flexible schema to store different types in same table
    ▪ Scenario: Rewards for user on every rental
    ▪ User info and rental info in same table
      – Both entities use PartitionKey == UserId
      – Prefix row key with type
      – User Entity RowKey = “”
      – Rental Entity RowKey = Rental-<Movie Name>
    ▪ Get user and rental rows for user id
      – Select * where PartitionKey = <UserId>
    ▪ Get user row for user id
      – Select * where PartitionKey = <UserId> && RowKey = “”
    ▪ Get rental rows for user id
      – Select * where PartitionKey = <UserId> && RowKey.StartsWith(“Rental”)
Data Modeling for Scale
Summarize

- What are my significant queries?
- What atomic transactions do I need?
- Utilize flexible schema
- Make query processing fast by doing some work upfront
  - Maintain Index Tables
  - Denormalize data
  - Let applications do the join or shape the data for joins
Summary

- Windows Azure Storage provides highly scalable, durable and available cloud storage system with strong consistency
- Pay for what you use model
- Abstractions - Blobs, Tables, Queues, Drives & Disks
- Windows Azure Tables is a NoSQL structured store that auto scales
- Patterns for designing large scale applications on Windows Azure Tables
References

- Storage team blog - http://blogs.msdn.com/b/windowsazurestorage/