Windows Azure Storage: A Highly Available Cloud Storage with Strong Consistency

Microsoft Research
Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyn Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, Leonidas Rigas

Presented by: M.P. Waldhorst, 1822004
Coming up...

- Windows Azure
- Data abstractions
- Architecture
  - Partition Layer
  - Stream Layer
- Load Balancing
- Recovery
- Consistency
- Throughput Evaluation
- Workload Profiles
- Conclusion
Windows Azure

- Windows Azure
  - Computation
    - Virtual Machines
  - Storage
    - Windows Azure Storage (WAS)

- Implemented independently
Data abstractions

* Different types of data:
  * Blobs
  * Files
  * Tables
  * Queues

* Access by a global namespace for all the data:
  * Example:
    http(s)://AccountName.<service>.core.windows.net/PartitionName/ObjectName
Focus on:

- Stream Layer
- Partition Layer
Object Table

- Massive table
  - Several petabytes
- Divided into RangePartitions
- All data types are implemented with these tables
Architecture: Partition Layer II

* Partition Manager:
  * Splitting Object Tables into RangePartitions
  * Distribution of RP to Partition Servers

* Partition Server:
  * Serves requests on RP’s
  * Only 1 server can service a RP at a time
Architecture: Stream Layer I

- Streams
- Extents
  - Immutable when sealed
- Blocks

Stream //foo

Pointer to Extent E1

B_{11} B_{12} \ldots B_{1x}

Extent E1 - Sealed

Pointer to Extent E2

B_{21} B_{22} \ldots B_{2y}

Extent E2 - Sealed

Pointer to Extent E3

B_{31} B_{32} \ldots B_{32}

Extent E3 - Sealed

Pointer to Extent E4

B_{41} B_{42} B_{43}

Extent E4 - Unsealed
Write at primary extent node
Replication using secondary nodes
Load Balancing

* **Stream Layer**
  * If an Extent node is busy it will tell the client (Partition Layer) to go to another replica

* **Partition Layer**
  * Uses metrics to determine load of different partition servers
    * (Transactions/second, CPU usage, network usage, data size...)
  * Offloads RangePartitions to less busy Partition server
Recovery

* Stream Layer
  * Intra-Stamp Replication
    * Primary/Secondary Extent
    * Replication time: as append happens
  * Partition Layer
    * Inter-Stamp Replication
      * Primary/Secondary storage stamp of an account
      * Replication time: ±30 seconds
* Disaster recovery might cause recent updates to be lost (last 30 seconds)
Consistency

- **Intra-Storage Stamp**
  - Replica’s are made consistent on each append operation in the stream layer
  - RangePartitions are only served by 1 server at a time

- **Inter Storage Stamp**
  - Lazy replication -> eventual consistency
Throughput Evaluation

- Evaluation setup:
  - Computation and Storage in same data center
  - Extra large VM’s
  - 1Gbps Network interface card
Throughput Evaluation

- Random single entity get and put operations on 100 GB Table
- Linear scaling
- Batch operations scale much better
  - Less roundtrips on the network
  - Less stream writes required
**Throughput Evaluation**

- Randomly get or put 4MB sized Blobs
- Scales almost linear
- Tempers of as we reach NIC limitations of 1Gbps
Very diverse uses of the system

<table>
<thead>
<tr>
<th></th>
<th>%Requests</th>
<th>%Capacity</th>
<th>%Ingress</th>
<th>%Egress</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blob</td>
<td>17.9</td>
<td>70.31</td>
<td>48.28</td>
<td>66.17</td>
</tr>
<tr>
<td>Table</td>
<td>46.88</td>
<td>29.68</td>
<td>49.61</td>
<td>33.07</td>
</tr>
<tr>
<td>Queue</td>
<td>35.22</td>
<td>0.01</td>
<td>2.11</td>
<td>0.76</td>
</tr>
<tr>
<td><strong>Bing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blob</td>
<td>0.46</td>
<td>60.45</td>
<td>16.73</td>
<td>29.11</td>
</tr>
<tr>
<td>Table</td>
<td>98.48</td>
<td>39.55</td>
<td>83.14</td>
<td>70.79</td>
</tr>
<tr>
<td>Queue</td>
<td>1.06</td>
<td>0</td>
<td>0.13</td>
<td>0.1</td>
</tr>
<tr>
<td><strong>XBox GameSaves</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blob</td>
<td>99.68</td>
<td>99.99</td>
<td>99.84</td>
<td>99.88</td>
</tr>
<tr>
<td>Table</td>
<td>0.32</td>
<td>0.01</td>
<td>0.16</td>
<td>0.12</td>
</tr>
<tr>
<td>Queue</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>XBox Telemetry</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blob</td>
<td>26.78</td>
<td>19.57</td>
<td>50.25</td>
<td>11.26</td>
</tr>
<tr>
<td>Table</td>
<td>44.98</td>
<td>80.43</td>
<td>49.25</td>
<td>88.29</td>
</tr>
<tr>
<td>Queue</td>
<td>28.24</td>
<td>0</td>
<td>0.5</td>
<td>0.45</td>
</tr>
<tr>
<td><strong>Zune</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blob</td>
<td>94.64</td>
<td>99.9</td>
<td>98.22</td>
<td>96.21</td>
</tr>
<tr>
<td>Table</td>
<td>5.36</td>
<td>0.1</td>
<td>1.78</td>
<td>3.79</td>
</tr>
<tr>
<td>Queue</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Windows Azure Storage provides:
- A single namespace to access data
- Strong consistency
- Disaster recovery
- Data access with different types of data scales

By splitting computation and storage and layering the system, we can provide strong consistency and availability that scales!