Enabling Native Integration of NoSQL HBase with Cloud Providers

Ankit Singhal, Wellington Chevreuil, Stephen Wu
Agenda

- TCO
  - Instance Sizing
  - Automatic scaling
  - Storage Optimization
- Storage Options
- Security Simplification
- Availability and Fault Tolerance
- Benchmark
- Case study
Considering Cloud Native

- Elasticity with increased efficiency
- Flexible and configurable Availability
- Cost Reduction

TCO
Instance sizing

- Per role type: Masters, RegionServers
- Per load (with RS Grouping)
  - System tables (meta, namespace, phoenix)
  - Use case specific load
TCO
Metrics based automatic scale up/down

● Read/Write latency/throughput based
● RPC latency based
● Compaction queue size
● Overall request load
● Region density
● Available cache space (Cloud Storage deployments)
TCO

Storage Optimizations

- Block vs Cloud Storage
- Ephemeral Storage for BucketCache
Storage Options

Cloud Storage

● Benefits
  ○ Cost efficient for large volumes
  ○ Available on major cloud providers
  ○ Decoupled compute from storage

● Challenges
  ○ Increased latency
  ○ HBase native compatibility
    ■ S3 lacks atomic rename (for now)
Mitigating latency

- File based bucket cache over ephemeral storage
  - Initial cache warmup required
  - Cache must be resilient to RS crashes/restarts
    - HBASE-27686, HBASE-27743
  - Region balancer must consider impacts to the cache
    - HBASE-27389

- Cloud provider specific tunings
  - Throttling, connection/thread pooling configs
  - Request hints (random/sequential)
Storage Options
Cloud Storage

● HBase S3 Integration
  ○ HBase originally designed for hierarchical file systems
    ■ HBase internal write operations are two phased:
      ● New files created on a temp dir
      ● Rename new files to final dir at commit time
    ■ Amazon S3 lacks atomic renames
      ● Requires locking the whole dir content
      ● Individual files rename calls to S3 (suboptimal)
Storage Options

Cloud Storage

Original Flow
Cloud Storage

● HBase S3 Integration
  ○ Redesign HBase to not (only) rely on renames
    ■ Store File Tracking: HBASE-26067
      ● Additional layer for tracking store files
      ● Write operations delegate file path decision to the tracking layer
      ● Configurable tracker implementations
        ○ Default: still uses renames
        ○ File Based Tracker: keeps list of valid files in meta files
Storage Options

Cloud Storage

SFT Flow

- flushCache
  - createWriter
  - createWriter
  - create
  - write
  - commitStoreFiles
  - commitStoreFiles
  - addStoreFiles
  - add
Storage Options

Cloud Storage

SFT Flow
(detailed)
Storage Options

Cloud Storage

● Limitations
  ○ No durable syncs (fsync/hflush)
    ■ WAL files require durable file sync to overcome crashes
    ■ Low latency critical for write performance

● Deploy HDFS for WAL
  ○ Separate WAL from data directory
  ○ WALs are temporary, so limited space usage
  ○ Use the instances attached disks for HDFS data
Storage Options

Block Storage

● Benefits
  ○ Higher performance *

● Challenges
  ○ More costly
  ○ Compute/Storage tightly coupled
Storage Options

Block Storage

- Block storage volumes attached to the cluster nodes
- HDFS deployed over the nodes
  - Both WALs and HBase data on HDFS
- Less cost efficient
  - Require additional/higher volumes attached to instances
  - HDFS replication factor requires more storage
  - Compute and Storage scaling together

- Higher Performance
  - Avg 5x better latency/throughput
## Storage Options

### Cloud vs Block

<table>
<thead>
<tr>
<th>Storage Type</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Storage</td>
<td>More cost efficient</td>
<td>Lower performance</td>
</tr>
<tr>
<td></td>
<td>Separate compute from storage:</td>
<td>Mitigated with bucket cache</td>
</tr>
<tr>
<td></td>
<td>- Independent scaling</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Data management and availability</td>
<td></td>
</tr>
<tr>
<td>Block Storage</td>
<td>Better overall performance</td>
<td>Higher costs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Couples compute and storage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Can’t scale independently</td>
</tr>
</tbody>
</table>
Security Simplification

Kerberos vs JWT

- Kerberos
  - HBase support since early versions
  - Strong client/server authentication (KDC based)
  - SASL for the RPC authentication
  - Requires clients to be known by the KDC
  - Complex to integrate with other services
Security Simplification

Kerberos vs JWT

- JWT
  - Token based authentication
    - Custom SASL auth provider (HBASE-23347)
    - HBase builtin works ongoing (HBASE-26553)
  - TLS for RPC (encrypts the token and further info)
    - HBASE-26666 recently added TLS for RPC support
  - Centralised authentication service
    - Easily reusable by clients/other services
Availability and Fault Tolerance

- HBase built-in capabilities
  - Master HA
  - Table sharding (Regions)
  - Meta region replica

![Diagram of HBase architecture](https://example.com/hbase-diagram.png)
Availability and Fault Tolerance (cont.)

- * Multiple Availability Zones (Multi AZ)
  - Master instances on different zones
  - RSes spread evenly throughout zones
  - Increases latency

*https://blog.cloudera.com/high-availability-multi-az-for-cdp-operational-database/
Availability and Fault Tolerance (cont.)

- Limitations
  - Cache reload (Cloud Storage deployments) when RSs crashes
    - HBASE-27313 Persist list of HFiles after prefetch
    - HBASE-27743 Enhancement for persistent cache
    - HBASE-27389 Cost function that considers bucket cache as part of the plan before region movement
  - Single physical region unavailability
    - Cross-region cluster replication for better RTO experience
Benchmark

YCSB Workloads

- Dataset size: 20 Billion rows = ~20TB
- Workload C: 100% Read
- Workload A: 50% read, 50% write
- Workload F: 50% read, 25% update, 25% read-modify-update
- Two common deployments, HDFS vs Cloud storage
Clusters definitions

- **AWS**
  - HDFS
    - 20 RSes m5.2xlarge storage with HDD
    - 2 Masters m5.8xlarge
  - S3
    - 20 RSes i3.2xlarge storage as S3
    - 2 Masters m5.2xlarge

- **Azure**
  - HDFS
    - 20 RSes Standard_D8_V3
    - 2 Masters Standard_D32_V3
  - ABFS
    - 20 RSes Standard_L8s_V2
    - 2 Masters Standard_D8a_V4
Benchmark

AWS with HDFS vs S3 (Higher is better)

![AWS-HBase-Throughput](image)

Throughput (ops/sec)

- WL A Throughput
- WL C Throughput
- WL F Throughput

Workloads:
- HDFS
- S3
Benchmark

AWS with HDFS vs S3 (Lower is better)

**AWS-HBase-Read Latency**

**AWS-HBase-Write Latency**

There is an outlier on P95+, where the P95 latency is 28.3 ms vs 3.3 ms.
Benchmark

Azure with HDFS vs ABFS (Higher is better)
Benchmark

Azure with HDFS vs ABFS (Lower is better)
### Benchmark

Average comparison against HDFS on Block Storage

<table>
<thead>
<tr>
<th>Workload Type</th>
<th>Latency S3 vs HDFS</th>
<th>Throughput S3 vs HDFS</th>
<th>Latency ABFS vs S3</th>
<th>Throughput S3 vs HDFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read only workload</td>
<td>47% lower</td>
<td>86% higher</td>
<td>48% lower</td>
<td>91% higher</td>
</tr>
<tr>
<td>50% Read, 50% Write</td>
<td>R: 52% lower</td>
<td>36% higher</td>
<td>R: 46% lower</td>
<td>66% higher</td>
</tr>
<tr>
<td></td>
<td>W: 28% higher</td>
<td></td>
<td>W: 35% lower</td>
<td></td>
</tr>
<tr>
<td>50% read, 25% update, 25% read-modify-update</td>
<td>R: 87% lower</td>
<td>66% higher</td>
<td>R: 48% lower</td>
<td>66% lower</td>
</tr>
<tr>
<td></td>
<td>W: 29% lower</td>
<td></td>
<td>W: 35% lower</td>
<td></td>
</tr>
</tbody>
</table>
Cost Case Study
Sample comparison for a 50TB read/write workload on AWS:

<table>
<thead>
<tr>
<th>50TB read/write workload</th>
<th>S3 with ephemeral</th>
<th>HDFS in block storage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of instances</strong></td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td><strong>Instance Types</strong></td>
<td>3 x m5.2xlarge</td>
<td>3 x m5.8xlarge</td>
</tr>
<tr>
<td></td>
<td>40 x i3.2xlarge</td>
<td>40 x m5.2xlarge</td>
</tr>
<tr>
<td><strong>Initial capacity with</strong></td>
<td>64TB ephemeral</td>
<td>190TB EBS volumes</td>
</tr>
<tr>
<td><strong>performance parity</strong></td>
<td>storage for</td>
<td>for HDFS</td>
</tr>
<tr>
<td></td>
<td>bucket cache</td>
<td></td>
</tr>
<tr>
<td><strong>Cost of instances</strong></td>
<td>$10,778.58</td>
<td>$7,467.51</td>
</tr>
<tr>
<td><strong>Cost of storage</strong></td>
<td>*$1,428.60</td>
<td>$9,216.00</td>
</tr>
<tr>
<td><strong>Total cost (monthly)</strong></td>
<td>$12,207.18 (-26.8%)</td>
<td>$16,683.51</td>
</tr>
</tbody>
</table>

*we estimated the amount of list/put/get requests to S3 is 50 millions per month, the actual cost may be slightly higher*
Speaker information

- Wellington Chevreuil
  - SW Engineer at Cloudera, HBase PMC

- Stephen Wu
  - SW Engineer at Cloudera, HBase PMC

- Specials thanks to OpDB Team @ Cloudera for all the contributions
  - Andor, Ankit, Sergey, Surbat, Peter, Shanmukha, Rahul and more people
CLOUDERA