A GPU-Powered Real-time Analytics Engine

Uber
Agenda

- Real-time analytics at Uber
- Leveraging GPU for real-time analytics
- AresDB Architecture and Features
- Learnings From GPU Programming
- Future Directions
Real-time analytics at Uber
Real-time Analytics Use Cases at Uber

- Rider eyeballs
- Dynamic Pricing
- Open car information
# Real-time Analytics Use Cases Categorization

<table>
<thead>
<tr>
<th></th>
<th><strong>Dashboards</strong></th>
<th><strong>Decision Systems</strong></th>
<th><strong>Ad hoc Queries</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dataset</strong></td>
<td>Subset</td>
<td>Subset</td>
<td>All data</td>
</tr>
<tr>
<td><strong>Ingestion Latency</strong></td>
<td>Seconds to Minutes</td>
<td>Seconds to Minutes</td>
<td>Minutes</td>
</tr>
<tr>
<td><strong>Query Pattern</strong></td>
<td>Well known</td>
<td>Well known</td>
<td>Arbitrary</td>
</tr>
<tr>
<td><strong>Query QPS</strong></td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Query Latency</strong></td>
<td>Sub seconds</td>
<td>Sub seconds</td>
<td>Minutes</td>
</tr>
<tr>
<td><strong>Target Users</strong></td>
<td>City OPS, Executives</td>
<td>Engineers (application developers)</td>
<td>Data Scientists, Analytics, City OPS</td>
</tr>
</tbody>
</table>
Mission of AresDB

- sub-sec level query latency
- second to min level ingestion latency
- High availability (4 9s)
- High data accuracy (3 9s)
- Uber scale and beyond

Build a fast, reliable and scalable analytics platform solution to power Uber’s Real-Time business intelligence
Leveraging GPU for Real-Time Analytics
AresDB: A GPU-Powered Real-time Analytics Engine

- High-efficiency storage
- Low-latency ingestion
- Sub-second query response time
- Feature set for real-time analytics
How is AresDB used at Uber

Ingestion (with upserts) < 1min

Query < 1 sec
The Problem: Time-series Analytics

Computing measures by dimensions on time series data

<table>
<thead>
<tr>
<th>request_at</th>
<th>city_id</th>
<th>fare</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-04-13 10:25</td>
<td>1</td>
<td>15.3</td>
</tr>
<tr>
<td>2017-04-13 11:10</td>
<td>1</td>
<td>7.5</td>
</tr>
<tr>
<td>2017-04-14 10:35</td>
<td>1</td>
<td>20.1</td>
</tr>
<tr>
<td>2017-04-14 11:40</td>
<td>5</td>
<td>12.1</td>
</tr>
<tr>
<td>2017-04-14 15:45</td>
<td>5</td>
<td>5.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>day(request_at)</th>
<th>city_id</th>
<th>sum(fare)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-04-13</td>
<td>1</td>
<td>22.8</td>
</tr>
<tr>
<td>2017-04-14</td>
<td>1</td>
<td>20.1</td>
</tr>
<tr>
<td>2017-04-14</td>
<td>5</td>
<td>17.7</td>
</tr>
</tbody>
</table>

computes
Why are GPUs well-suited?
GPU vs CPU

Intel® Xeon® Processor E5-2620 v3

12 Threads
59GB/s
500 GFIOPS

Tesla P100

3584 CUDA cores
549G/s
9.3 TFIOPS
GPU vs CPU

Theoretical GFLOP/s at base clock

- NVIDIA GPU Single Precision
- NVIDIA GPU Double Precision
- Intel CPU Single Precision
- Intel CPU Double Precision

2003 2005 2007 2009 2011 2013 2015
Hardware Storage Choices

CPU
- Control
- Cache

DRAM (256G)

GPU

ALU

ALU

59GB/s

16GB/s

PCle

DRAM (12G)

549G/s
## Hardware Storage Choices

<table>
<thead>
<tr>
<th></th>
<th>Capacity</th>
<th>Bandwidth to GPUs</th>
<th>Ingestion Method</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDD</td>
<td>8TB</td>
<td>100MB/s</td>
<td>File writing similar to traditional databases</td>
<td>Unable to feed data fast enough to fully utilize GPUs</td>
</tr>
<tr>
<td>SSD</td>
<td>4TB</td>
<td>600MB/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVMe</td>
<td>2TB</td>
<td>3GB/s</td>
<td></td>
<td>And also expensive</td>
</tr>
<tr>
<td>Host Memory</td>
<td>256GB</td>
<td>15GB/s per side; 30GB/s two sides</td>
<td>Memory writing</td>
<td>Limited by PCIe bandwidth</td>
</tr>
<tr>
<td>GPU Memory</td>
<td>12GBx8</td>
<td>500GB/s on the same GPU; 15GB/s across GPUs</td>
<td>Sharding across multiple GPUs; Complex memory writing</td>
<td>Tight coupling of storage and computation; ingestion is challenging</td>
</tr>
</tbody>
</table>
AresDB Architecture and Features
AresDB Architecture: Single Instance

- **Query**
- **Ingestion**
- **Schema**

**API**

**Memory**
- MemStore
- MetaData

**Disk**
- DiskStore
- MetaStore

**GPU**
Fact/Dimension Table

- **Fact table**
  - Facts about a business process
  - Each record associated with an event time (grows with time)
  - E.g. trips, orders, ...

- **Dimension table**
  - Descriptive attributes/dimensions
  - E.g. product catalogs, cities, ...
Star Schema

- **Dimension 1**
  - Primary Key
  -...

- **Dimension 2**
  - Primary Key
  -...

- **Dimension 3**
  - Primary Key
  -...

- **Dimension 4**
  - Primary Key
  -...

- **Facts**
  - EventTime
  - Primary Key
  - Foreign Keys...
  -...

Feature Highlights

- In-Memory Columnar Storage
- Real-time upserts
- GPU powered query engine

Analytical Query Feature Set
- Time zone, Time Filter, Time Bucketization
- Geospatial analytics
- Fact/dimension table joins
- Hyperloglog
Columnar Storage: uncompressed

<table>
<thead>
<tr>
<th>city_id</th>
<th>Value Vector</th>
<th>NULL Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Los Angeles</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>New York City</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>NULL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>San Francisco</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Paddings</td>
<td></td>
<td>1 Byte</td>
</tr>
</tbody>
</table>

2 Bytes

1 Byte
Columnar Storage: compressed

<table>
<thead>
<tr>
<th>Value Vector</th>
<th>Null Vector</th>
<th>Count Vector</th>
<th>Value Vector</th>
<th>Null Vector</th>
<th>Count Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>x1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SF</td>
<td>x3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NYC</td>
<td>x2</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LA</td>
<td>x2</td>
<td>12</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

One to Many

<table>
<thead>
<tr>
<th>city_id</th>
<th>status</th>
</tr>
</thead>
</table>
| SF      | completed x2, canceled x1
| NYC     | completed x2, canceled x1
| LA      | completed x2, canceled x1

Note: The diagram shows the distribution of city IDs and statuses in a columnar storage format, with compressed data representation.
Columnar Storage: fact table

Fact Table

- Archive Store (compressed)
- Live Store (uncompressed)
- Archive Files (compressed)
- Redo Logs

- Mature Data
- Most Recent Data

- MemStore
- DiskStore
- Preloading / On Demand Loading

- Archive: sort, compress, merge
- Recovery
- Ingestion
- Backfill
Columnar Storage: dimension table

Dimension Table

MemStore

DiskStore

Load

Snapshot

Live Store (uncompressed)

Ingestion

Snapshot

Redo Logs

Recovery
Feature Highlights

- In-Memory Columnar Storage
- Real-time upserts
- GPU powered query engine
- Analytical Query Feature Set
  - Time zone, Time Filter, Time Bucketization
  - Geospatial analytics
  - Fact/dimension table joins
  - Hyperloglog
Real-time upserts: ingestion flow
Real-time upserts: deduplication

Live Store

- Primary Key Hash Index:
  - 789-56-1234 -100:0
  - 123-45-6789 -99:12
  - 012-34-5678 -100:30

- Batch -100
- Batch -99

Columnar Vector
Empty Space
Real-time upserts: archiving

Archive Store Version 1512090000

Batch 17500
2017-11-30
Version [1512090000, 7]

Batch 17501
2017-12-01
Version [1512090000, 5]

Patch From Live Store
From: 1512090000
To: 1512120000

Batch 17500
2017-11-30
Version [1512090000, 7]

Batch 17501
2017-12-01
Version [1512090000, 5]

Archive Merge

Sort And Compressed Columnar Vector

Archive Store Version 1512120000

Batch 17500
2017-11-30
Version [1512090000, 7]

Batch 17501
2017-12-01
Version [1512090000, 0]
Real-time upserts: event timeline

Ingestion

- Past
- cut-off
- Now
- Event Time

Write to backfill queue
Write to live store
Real-time upserts: event timeline

Query

- Past
- cut-off
- Now
- Event Time

Read from Archive Store
Read from Live Store
Real-time upserts: event timeline

Archiving

Past  |  Event Time
-----|---------
- cut-off  |  new cut-off  |  Now

Archiving interval  |  Archiving Delay

Records | Records Being Archived
Already Archived  |  Archived
Real-time upserts: event timeline

Archiving

Past

Cut-off

Now

Event Time

Records Already Archived
Feature Highlights

- In-Memory Columnar Storage
- Real-time upserts with deduplication
- GPU powered query engine
- Analytical Query Feature Set
  - Time zone, Time Filter, Time Bucketization
  - Geospatial analytics
  - Fact/dimension table joins
  - Hyperloglog
Query Engine

High level architecture
Data Feeding

CPU
- Control
- Cache

ALU
- ALU

DRAM (256G)

GPU

DRAM (12G)

PCIe
- 16GB/s

59GB/s

549GB/s
Data Feeding

Partitioned Data

Live Batches

Archive Batches

2017-11-13

2017-11-12

2017-09-13
Data Feeding

Prefilter

Sort Columns

<table>
<thead>
<tr>
<th>city_id</th>
<th>status</th>
<th>vvid</th>
<th>fare</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>1</td>
</tr>
</tbody>
</table>

One to Many

Matched Prefilters

city_id = 1
status = 0
vvid >= 20
Data Feeding

Pipelining

Stream 0
Batch 0 → Batch 0 → Batch 2 → Batch 2 → Batch 4 → Batch 4

Stream 1
Batch 1 → Batch 1 → Batch 3 → Batch 3 → Batch 5

Copy data from host to device
Query execution
Query Execution

Execution Stages

Filters Evaluation → Dimension Expressions Evaluation → Measure Expressions Evaluation → Sort → Reduction
One operator per kernel on non-leaf nodes
- Each leaf node is one of
  - column/constant
- Non-root, non-leaf node
  - kernel launch
  - output to intermediate buffer
- Root node
  - Kernel launch
  - Write to output buffer
- E.g., request_at - request_at % 86400
Query Execution

Aggregation (Sort and Reduction)

Result from previous batches

Expression Evaluation Output

Dimension Vector

Measure Vector

Sort by dimensions

Reduce by dimensions

Final results
Device Resource Management

Query 1
Query 2
Query 3
Query 4
Query 5
Query 6

GPU1
GPU2
GPU3
GPU4

Resource Estimation

Query 6
Feature Highlights

- In-Memory Columnar Storage
- Real-time upserts with deduplication
- GPU powered query engine
- Feature set for analytical queries
  - Time zone, Time Filter, Time Bucketization
  - Fact/dimension table joins
  - Geospatial analytics
  - Hyperloglog
SELECT Count(''),
Unix_timestamp(Convert_tz(Concat(Date_format(Convert_tz(From_unixtime(((driver_info.first_active_at) -
(driver_info.first_active_at) % 900000) / 1000), 'GMT', 'America/Los_Angeles'), '%Y-%m-%d %H:'),
Lpad(15*Floor(Minute(Convert_tz(From_unixtime(((driver_info.first_active_at) -
(driver_info.first_active_at) % 900000) / 1000), 'GMT', 'America/Los_Angeles'))/15), 2, '0')), 'America/Los_Angeles', 'UTC')) AS time_dimension, driver_info.flow_type
FROM driver_info
WHERE driver_info.first_active_at >=
1534810500000
AND driver_info.first_active_at <
1534813200000
GROUP BY 2.3

V.S.

```json
{
  "table": "driver_info",
  "measures": [
    {
      "sqlExpression": "count(*)"
    }
  ],
  "dimensions": [
    {
      "alias": "ts",
      "sqlExpression": "first_active_at",
      "timeBucketizer": "day"
    },
    {
      "sqlExpression": "flow_type"
    }
  ],
  "timeFilter": {
    "column": "first_active_at",
    "from": "7 days ago"
  },
  "timezone": "America/Los_Angeles"
}
```
Analytical Query Features

- Fact/Dimension Table Join
  - E.g. trips.city_id = cities.id

- Hyperloglog Cardinality Estimation
  - countDistinctHLL(driver_id)
  - Dedicated hll column

- Geospatial analytics
  - GeoPoint, GeoShape
  - GeoIntersect(point, shape)
Learnings from GPU Programming
Learnings from GPU Programming

- Maximize parallelism
- Optimize memory access
- Maximize arithmetic intensity
- Reduce data transfer between GPU/CPU
- Profile, profile, profile
Maximize Parallelism

- Partition your computation to keep the GPU multiprocessors equally busy
  - Many threads, many thread blocks
  - E.g. Inclusion test for a point and polygons
    - One shape per thread vs one edge per thread
- Keep resource usage low enough to maximize occupancy
  - Register, shared memory
  - Careful design of data structure
    - Use less wide data type
      - Int64 -> uint32
    - Reuse memory space
      - Union
    - Passing offsets instead of pointers
Optimize Memory Access

- Coalesced vs. non-coalesced = order of magnitude
- Global/Local device memory
- Shared memory
- Constant memory

retrieved date: 01/10/2019
Maximize Arithmetic Intensity

- GPU spends its transistors on ALUs, not memory
- Sometimes it’s better to recompute than to cache
- Do more computation on GPU instead of transferring back to CPU
Minimize CPU/GPU Transfers

- Group transfers
- Overlapping data transfers and computation
  - Async and stream api
  - Stream = sequence of operations that execute in order on GPU
  - Pipeline execution
- Pinned memory vs. pageable memory
Profiling GPU Program NVVP

- Nvidia visual profiler
- Unified CPU/GPU timeline
- Automated performance analysis
- Guided application analysis
NVVP cont’d
Future Directions

- Beyond single instance
  - Sharding
  - Replication
- Ease of adoption
  - SQL interface
  - Native Kafka support
- More query features (eg. fact to fact table joins)
- Query engine optimizations (eg. GPU memory caching)
- Grow AresDB together with the community
Questions?

Tech blog: https://eng.uber.com/aresdb/

Git repo: https://github.com/uber/aresdb

Questions: email uberopen@uber.com

Follow our Facebook page:
www.facebook.com/uberopensource
Thank you

Questions: email uberopen@uber.com

Follow our Facebook page:
www.facebook.com/uberopensource