BigQuery vs. Cassandra

DBMS Implementation - Term Project
Winter ‘21 - CS587

Aaron Hudson
Travis McGowan

Benchmark Approach and Goals

- System Choices:
    - Wanted to explore systems new to us
    - Cassandra and BigQuery are very different from one another
    - Both effective for handling very large amounts of data
- Benchmark Approach:
    - Focus on clarity and simplicity with experimental design due to how fundamentally different the systems are
    - Include testing very large amounts of data
- Goals:
    - Determine cases where each DBMS shines
    - Explore areas each system is different and see how that impacts performance

BigQuery

- No Indices
- Made for BIG data
- Simple Materialized Views
- Multiple ways to load data
- Save and schedule queries
- Almost no customization options

Cassandra

Cassandra by Evelyn de Morgan (1898, London); Cassandra in front of the burning city of Troy (https://en.wikipedia.org/wiki/Cassandra)

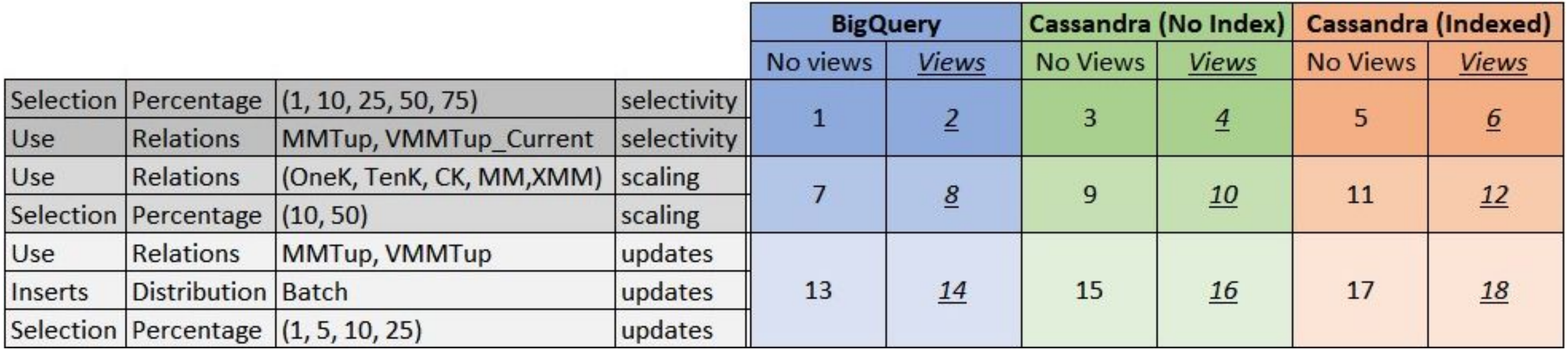
- No Joins
- NoSQL System (CQL)
- Indices: primary and secondary
- Employs a cluster of nodes to store data
- Performance is heavily driven by configuration

Testing Framework

4 Test Sets Comparing 3 Systems:

1. Selectivity
   a. Percentage & Non/View
2. Scaling
   a. Size & Selectivity
3. Tuple Insertion
   a. Size & Non/View
4. Views

All Testing Performed on Google Cloud Platform Virtual Machines



				BigQuery		Cassandra (No Index)		Cassandra (Indexed)	
				No views	*Views*	No Views	*Views*	No Views	*Views*
Selection	Percentage	(1, 10, 25, 50, 75)	selectivity	1	*2*	3	*4*	5	*6*
Use	Relations	MMTup, VMMTup_Current	selectivity						
Use	Relations	(OneK, TenK, CK, MM,XMM)	scaling	7	*8*	9	*10*	11	*12*
Selection	Percentage	(10, 50)	scaling						
Use	Relations	MMTup, VMMTup	updates	13	*14*	15	*16*	17	*18*
Inserts	Distribution	Batch	updates						
Selection	Percentage	(1, 5, 10, 25)	updates						

1. Scaling Experiment

- Designed to compare how the systems handle varying data sizes (1k, 10k, 100k, 1mm, 10mm)

- Used queries with 10% and 50% selectivity

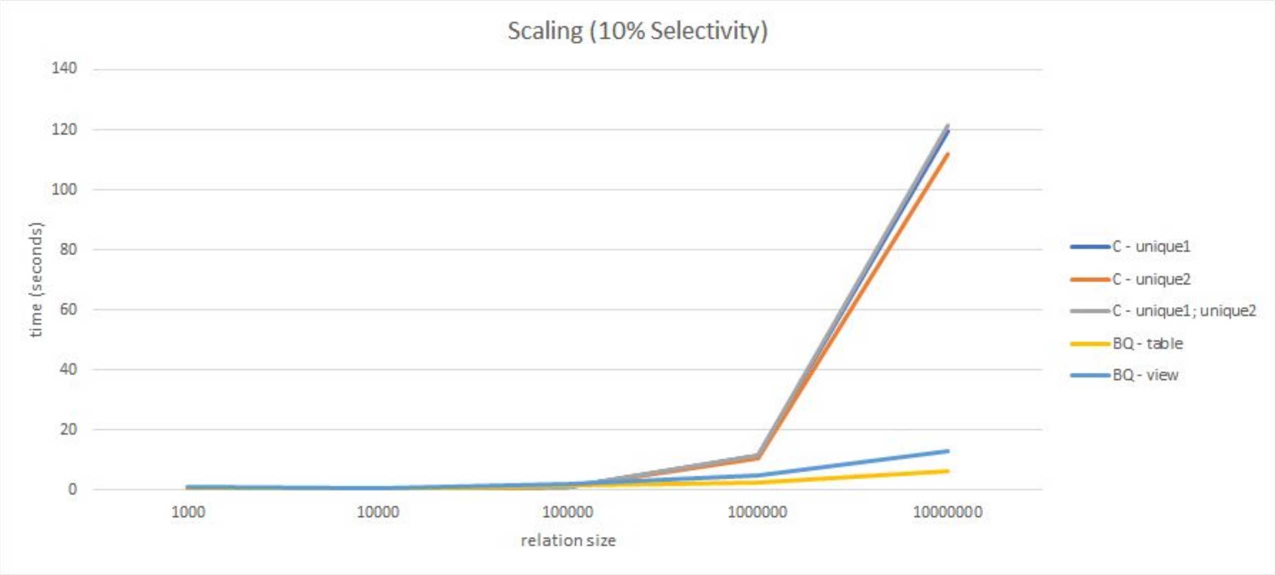
- General Query Format:

```
SELECT COUNT(unique2)
FROM relation_to_test
WHERE unique2 BETWEEN selectivity_range;
```

(Cassandra CQL doesn't have BETWEEN - used less than and greater than to replicate)

- Expected results:
  - As the relation size increases, BigQuery and Cassandra (no index) will degrade faster
  - Cassandra with an index will remain stable

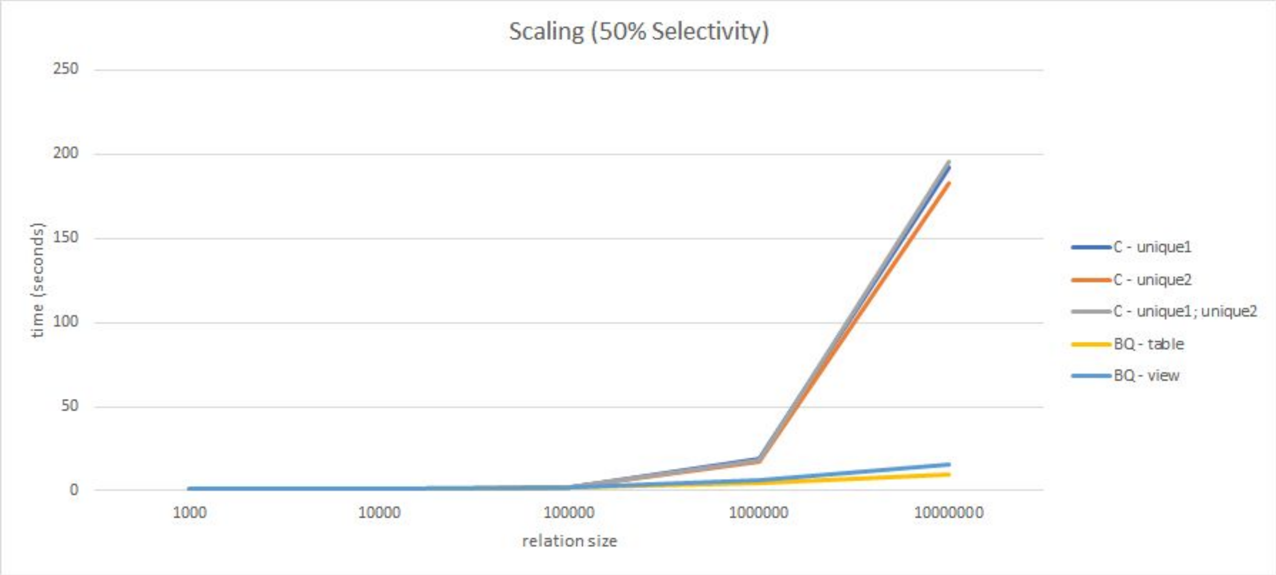
1. Scaling (10%)



Size	1 k	10 k	100 k	1 mm	10 mm
C -U1	**0.03**	**0.15**	**1.20**	**11.68**	**119.34**
C - U2	**0.02**	**0.12**	**1.08**	**10.77**	**111.71**
C - U1/2	**0.04**	**0.14**	**1.17**	**11.73**	**121.37**
BQ - T	**0.24**	**0.35**	**1.45**	**2.63**	**6.56**
BQ - V	**1.17**	**0.90**	**1.97**	**4.78**	**13.30**

- BigQuery performs better on large relations
- BigQuery performs quite a bit better than this if you allow the results to come from the cache
- Relations of 1mm seems to be the beginning pivot point for Cassandra VM

1. Scaling (50%)



Size	1 k	10 k	100 k	1 mm	10 mm
C -U1	0.05	0.32	2.18	18.76	192.65
C - U2	0.06	0.25	2.19	17.67	182.71
C - U1/2	0.04	0.24	2.23	18.56	195.90
BQ - T	0.27	1.47	1.82	4.29	9.47
BQ - V	0.99	1.61	2.41	6.25	15.31

- Cassandra performance stays relatively stable until 100k records (as compared to 10% scaling)
- BigQuery performance noticeably slows around 10k

1. Scaling Experiment - Conclusions

- BigQuery:
  - Performance remained stable as relations increased in size

- Cassandra:
  - Performance remained stable from 1k - 100k relation sizes
  - At 1 million and beyond performance dropped significantly
  - Likely due to data being spread across the cluster of nodes ineffectively

- Actual Results (in contrast with Expected):
  - As the relation size increases, BigQuery remained stable
  - Cassandra with an index performed better than without one, but still had considerable performance degradation after 1 million tuples

2. Selectivity Experiment

- Designed to compare how the systems handle queries of increasing selectivities (1%, 10%, 25%, 50%, 75%)

- Used relation with one million tuples for all queries

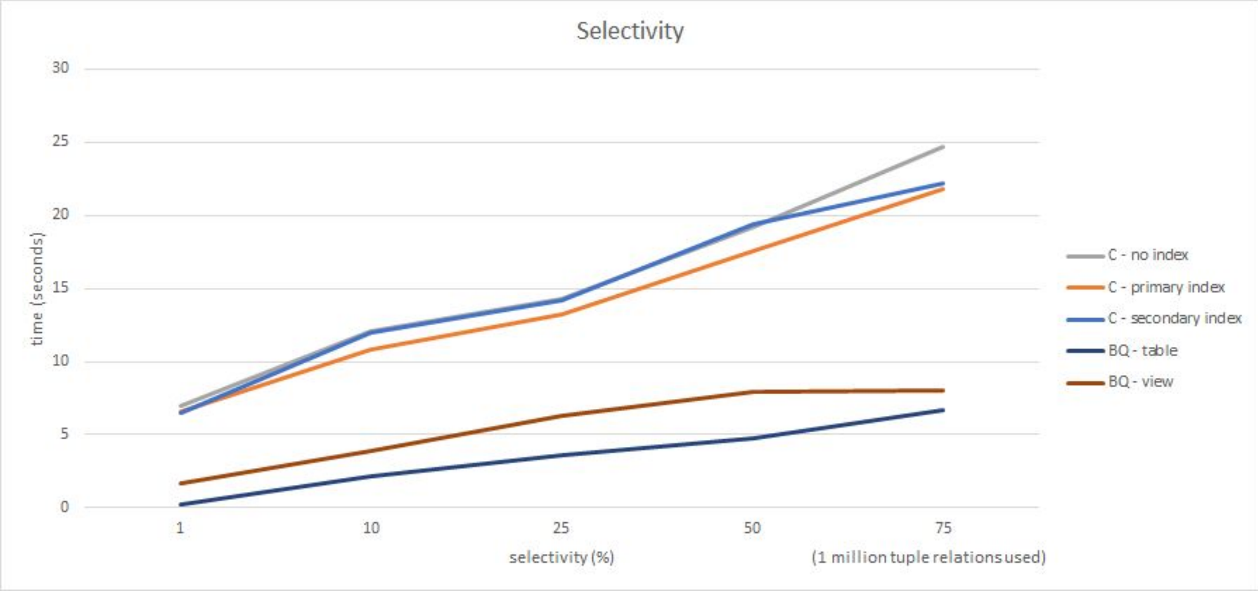
- General Query Format:

```
SELECT COUNT(unique2)
FROM MMTup
WHERE unique2 BETWEEN selectivity_range;
```

(Cassandra CQL doesn’t have BETWEEN - used less than and greater than to replicate)

- Expected results:
  - At higher selectivities BigQuery and Cassandra without an index will perform better (10% rule)
  - At lower selectivities, Cassandra with an index will perform better

2. Selectivity



Selectivity	1%	10%	25%	50%	75%
C - no index	7	12.10	14.3	19.25	24.74
C - primary index	6.5	10.80	13.20	17.53	21.81
C - secondary index	6.5	12	14.23	19.41	22.24
BQ - table	0.2	2.11	3.61	4.72	6.67
BQ - view	1.6	3.92	6.26	7.93	8.07

Cassandra Performance:

- Overall much worse than BQ
- Primary index performed best
- Secondary index made little difference except at 75% selectivity

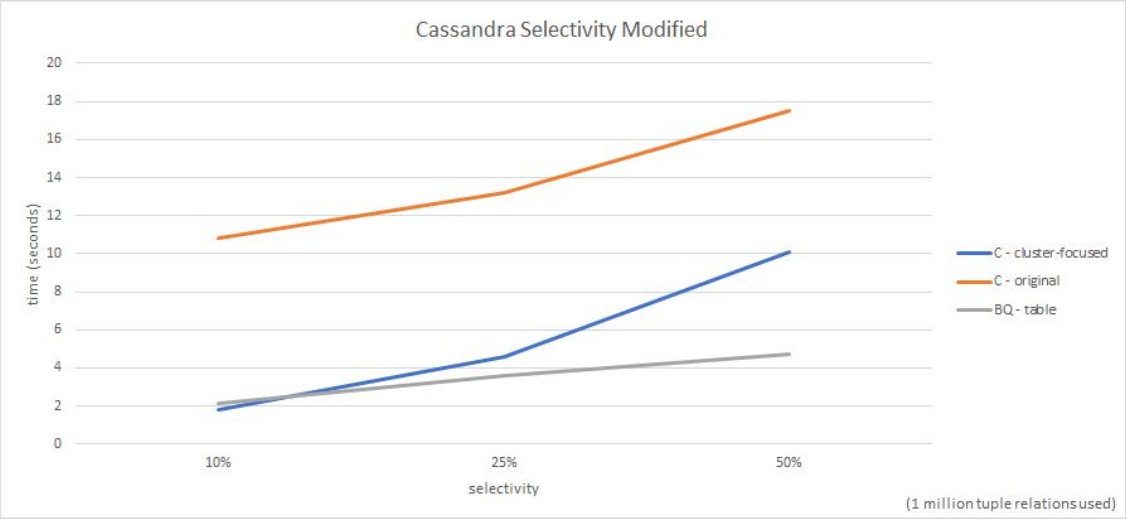
BigQuery Relations:

- Performed significantly better
- Use of materialized view hindered performance

Improving Cassandra Performance

- First attribute of Primary Key acts as a Partition Key (determines how tuples are organized in nodes across the cluster)
- Goal is to query data in a way that stays within a node
- Original queries not great for this
  - Required ALLOW FILTERING setting - visited multiple nodes
- Need to build database conscientiously
  - How data will be stored across the cluster
  - How data will be queried

2. Selectivity - Cassandra Modified



Selectivity	10%	25%	50%
C - cluster-focused	1.78	4.58	10.06
C - original	10.80	13.20	17.53
BQ-T	2.11	3.61	4.72

- Original relations had Partition Keys of unique1 or unique2
- Cluster-focused relations have Partition Keys on ten, four, and two depending on desired selectivity
- Performance closer to BQ
  (50% selectivity possibly requires visiting multiple nodes, which indicates the non-linear increase)

2. Selectivity Experiment - Conclusions

- BigQuery:
    - Consistent performance across all selectivities

- Cassandra:
    - Significantly worse performance with partition key on unique1/unique2
    - Performance more in line with BigQuery with partition key on ten/four/two
    - Likely impacted by how data is spread across cluster, and how many nodes must be visited to perform query

- Actual Results (in contrast with Expected):
    - At higher selectivities, BigQuery outperformed Cassandra significantly
    - At 10% selectivity, Cassandra with partition key on ten/four/two performed best, but was outperformed by BigQuery for higher selectivities

Overall Conclusions

- BigQuery is dominant and EASY
- Cassandra requires careful implementation
  - Partition Key choice impacts spread of data across cluster
  - Desired queries should utilize the clustering effectively
- BigQuery is cheap, while Cassandra requires persistent disk that can be costly in GCP
- Cassandra’s optimal performance likely not achieved

Lessons Learned

- Comparisons between systems requires nuance, and brute-force testing doesn’t always tell the entire story

- Cassandra punishes the ill-informed
  - Requires intimate knowledge of how it functions to bring out its power

- BigQuery is a great option that requires little investment for big returns

(https://fictionphile.com/cassandra-of-troy/)

Appendix

3. Tuple Insertion Experiment

- Designed to test the performance of each database when adding tuples to already existing relations

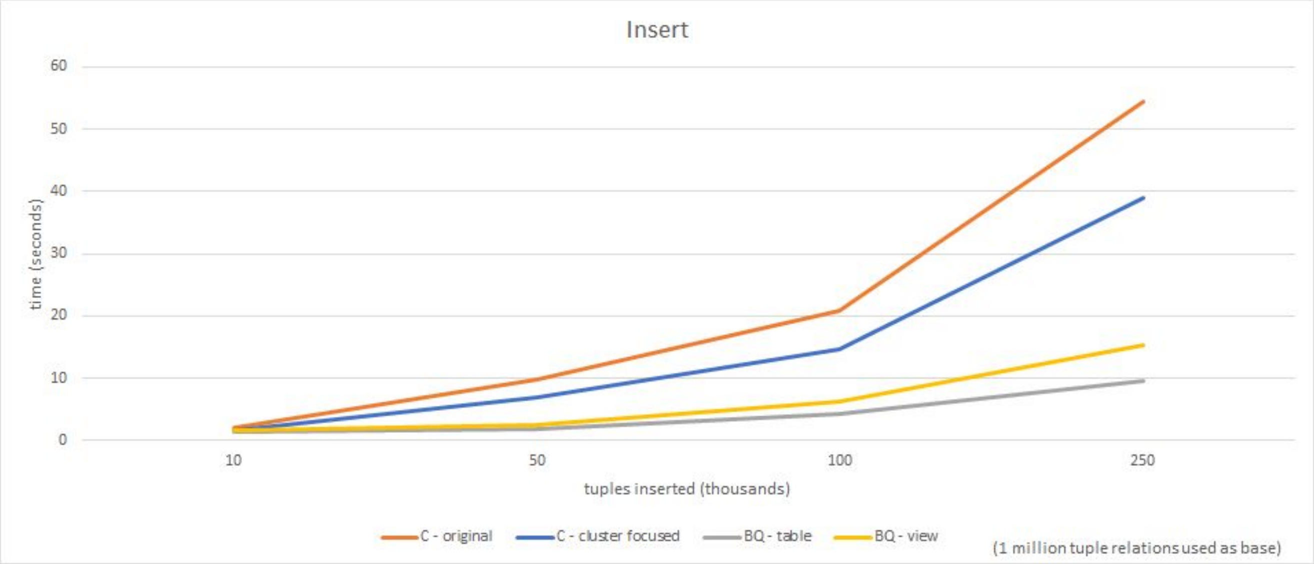
- Various amounts of tuples (10k, 50k, 100k, 250k) added to relations of 1 million tuples

- General Query Format:

```
COPY relation_name (attributes)
FROM 'filename.csv' WITH HEADER = true;
```

- Expected results:
  - BigQuery will have better performance, as there are no indices to keep updated
  - Cassandra will be slower because there is more internal upkeep with adding tuples (ie: partitioning data, maintaining indices, etc.)

3. Tuple Insertion Experiment



Tuples Inserted	10 k	50 k	100 k	250 k
C - original	2.14	9.77	20.87	54.41
C - cluster focused	1.59	6.98	14.75	39.07
BQ- table	1.47	1.82	4.29	9.47
BQ - view	1.61	2.41	6.25	15.31

- Inserting tuples into BigQuery relation with view was slightly less performant than without view
- Cassandra had similar performance to BigQuery starting out, but deviated more and more as the amount of tuples inserted grew
- Cassandra was faster inserting into the relation that was built with partitioning in mind

3. Tuple Insertion - Conclusions

- BigQuery
    - More stable as the number of tuples to insert increased
    - Likely due to not having as much internal maintenance to take care of during insertion (such as maintaining indices, etc).

- Cassandra:
    - Worst performance when inserting tuples into the original relation partitioned on unique2
    - Better performance inserting tuples into the cluster focused relation partitioned on ‘ten’ attribute
    - Worst case likely had to add all tuples to the same node (since organized by unique2)
    - Better case likely added them more evenly distributed across all nodes in the cluster (since organized by ten), allowing tuples to be added to different nodes concurrently

- Actual Results (in contrast with Expected):
    - BigQuery did have better performance, as expected
    - This is likely attributed to less internal maintenance for BigQuery as opposed to Cassandra (which must partition the data and maintain any indices)

4. Views Experiment

- Designed to test the performance impact using views hason each database

- Used queries with varying selectivities: 10%, 25%, 50%

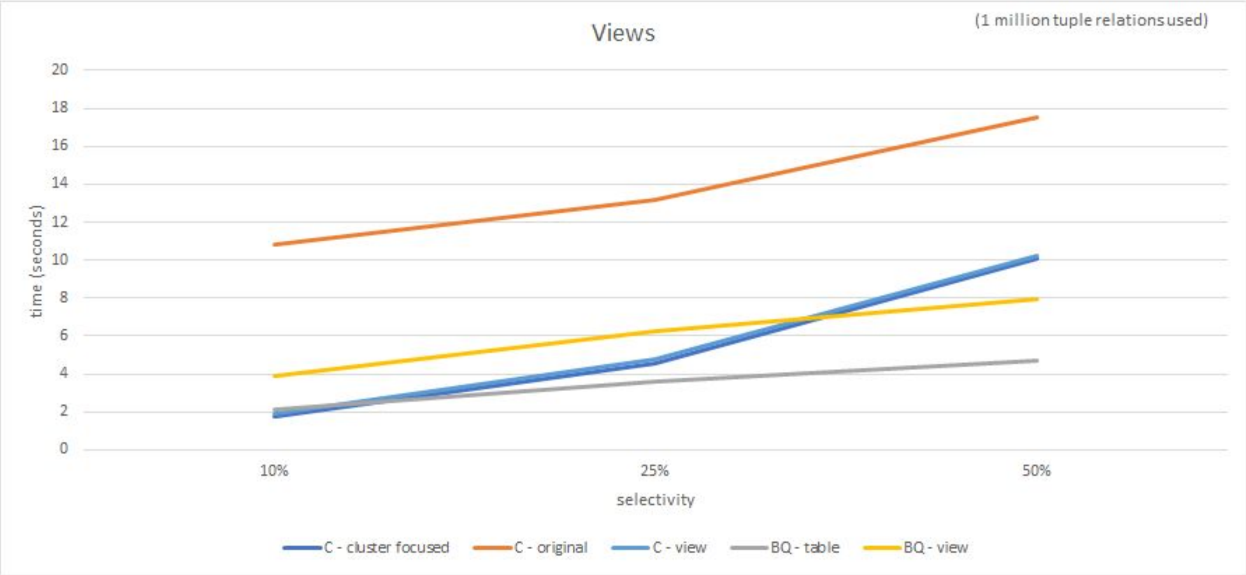
- General Query Format:

  ```
  SELECT COUNT(unique2)
  FROM relation_to_test
  WHERE attribute = 0;
  ```

  (attribute = ten, four, two depending on selectivity)

- Expected results:
  - Views will improve performance across the board, but will have greater impact as the relation size grows

4. Views



Selectivity	10%	25%	50%
C - cluster focused	1.78	4.58	10.06
C - original	10.80	13.20	17.53
C - view	1.93	4.79	10.26
BQ - table	2.11	3.61	4.72
BQ - view	3.92	6.26	7.93

- BigQuery view hinders performance slightly compared to no view
- Cassandra materialized view improves performance significantly from original relation, and is very comparable to the updated clustered focused relation performance

4. Views Experiment - Conclusions

- BigQuery:
  - Worse performance when querying the view than the relation itself
  - Likely due to overhead required by the view itself
- Cassandra:
  - Much better performance achieved when querying the materialized view than achieved with the original relation partitioned on ‘unique2’
  - Materialized view lets you create a version of the relation partitioned on another attribute, which explains why performance achieved matches the performance of the cluster focused relation
- Actual Results (in contrast with Expected):
  - Views did not improve performance of BigQuery, but did improve performance of Cassandra
  - Views didn’t make a noticeably greater difference in queries with higher selectivity

Changes to Experiments From Original Design

- Experiment 3: Inserting Tuples into Relations
  - Originally was focused on batch updates
  - Switched, due to limitations encountered in BigQuery, to focus on individual queries that inserted differing numbers of tuples into relations of 1 million tuples
  - Allowed us to attempt to compare impacts of behind the scenes maintenance on performance (such as maintaining indices)

- Experiment 4: Views
  - Originally was designed running the same query on relations of increasing sizes
  - Switched to
    - utilize a consistent 1 million tuple relation
    - varied the selectivity
  - We felt this change would be more interesting to see how views impact performance as the number of tuples returned increases