OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
INF FAKULTÄT FÜR INFORMATIK

Tiles and Non-Relational Models

Shreeraksha Achutha
Taranpreet Kaur
Madhuri Nagaraj Hebbale
Sindhuja Madabushi

Supervisor: Gabriel Campero
July 10th, 2017

Structure

1. Team Roles

2. Topic Introduction

- Motivation/Problem Definition
1. Our project proposal
- Background
1. Tile based Architecture – Physical and Logical Tiles
2. NOSQL Models – Key Value Stores and Document Stores
- Conceptual Model Implementation
1. Key Value Stores – Mapping to Tiles
2. Document Stores – Mapping to Tiles
- Evaluation Set up and Results
- Conclusion
- Future work

3. Project Progress

Team Roles for M4 :-

Shreeraksha Achutha
Program: Data and Knowledge Engineering
Main tasks: Literature research, paper presentation

Taranpreet Kaur
Program: Digital Engineering
Main tasks: Literature research, paper presentation

Madhuri Nagaraj Hebbale
Program: Data and Knowledge Engineering
Main tasks: Literature research, paper presentation

Sindhuja Madabushi
Program: Data and Knowledge Engineering
Main tasks: Literature research, paper presentation

Motivation – Our project proposal

- Majority of the data – *unstructured or semi -structured*. An increasing need to introduce effective storage mechanisms.
- Relational databases enforce *ACID properties*, strong transactional consistency and fixed schema.
- *NOSQL models* come to the rescue – Flexible schema, no enforced Integrity Constraints, performance benefits and scalability.
- *Tile Based Architecture* - FSM oriented DBMS architecture which combines the benefits of both NSM and DSM.
- *Main Idea* - Column stores have been used for NOSQL databases. We want to evaluate experimentally the potential of Tile based architecture to support NOSQL models.

Tile Based Architectures for Data layout

Andy Pavlo

- Tile based architecture is FSM (flexible storage model) oriented DBMS architecture (expectations of OLTP + OLAP in same database).
- FSM is a storage model that uses the benefits of NSM(n-ary storage model) and DSM(Decomposition storage model).

Logical tile:

Joy Arulraj

- Is an abstraction for the query engine (where all the primitive operations act over Logical tiles).
- Hides the way that the data is stored (represents an offset to a physical tile, or a set of them).
- Performance benefits.

Tile Based Architectures for Data layout

Physical Tile: The actual data is stored in physical tiles.

ORDERKEY	CUSTKEY	ORDERSTATUS	TOTAL	ORDERDATE	PRIORITY
1	3691	O	194029.55	02-01-199	-LOW
2	7801	O	60951.63	01-12-1996	1-URGENT
3	12332	F	247296.05	14-10-1993	5-LOW
4	13678	O	53829.87	11-10-1995	-LOW
5	4450	F	139660.54	30-07-1994	5-LOW
6	5563	F	65843.52	21-02-1992	4-NOT SPECIFIED

TILE-T-1

TILE-T-2

TILE-A-1

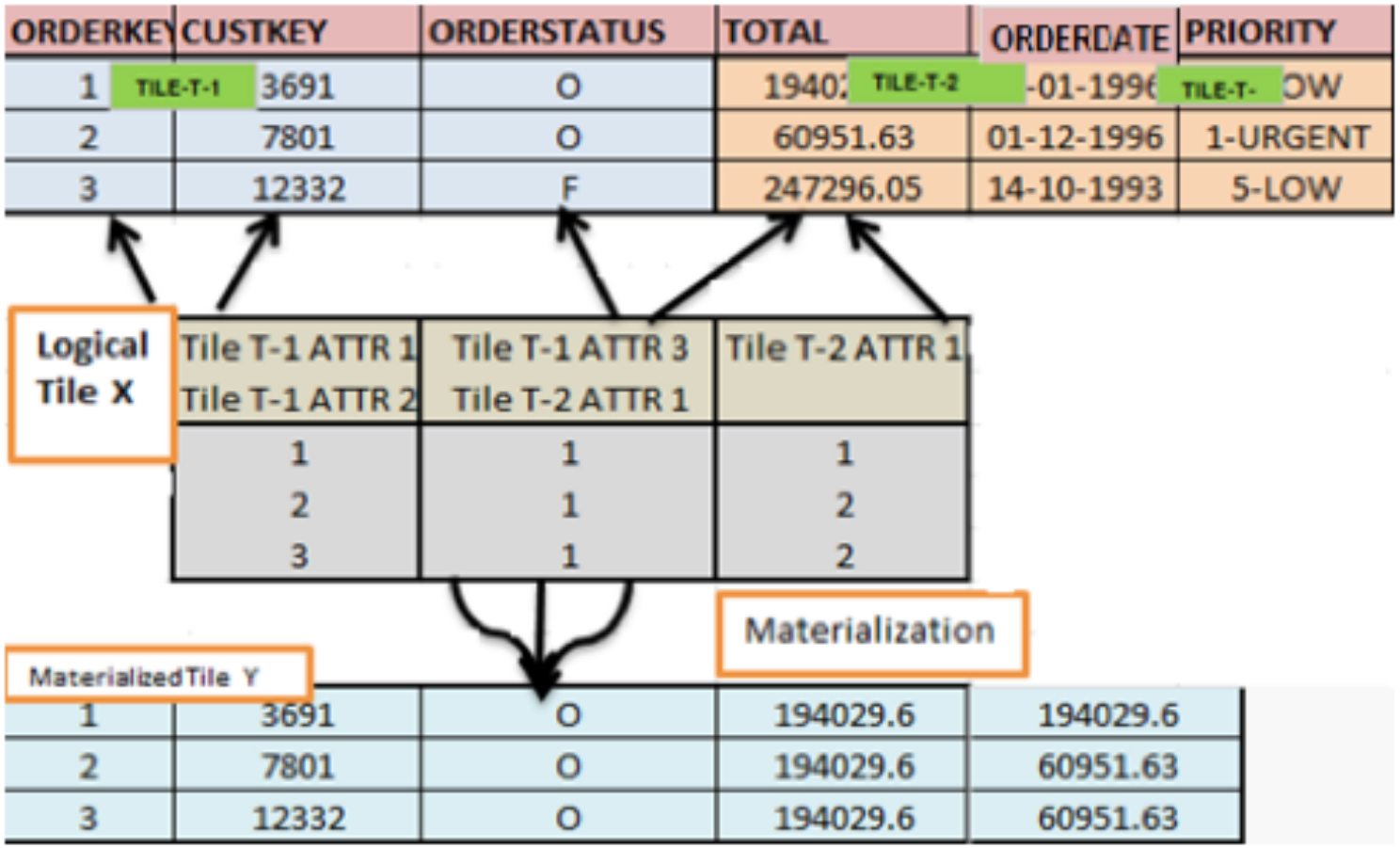
TILE-A-2

TILE-A-3

Tile-Group 'T'

Tile-Group 'A'

Logical and Physical Tiles :



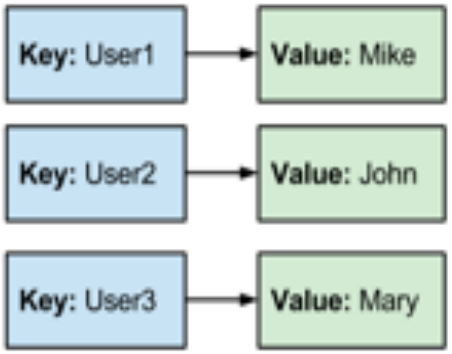
Data Models :

A data model is an abstract model that organises elements of data and standardises how they relate to one another and to properties of the real world entities. Examples : RDBMS, KVS, DS, Graphs

Features	Relational Models	DS	KVS	KVCS	PG	RDF
Rigid Schema	Yes	-	-	-	-	-
Flexible Schema	-	Yes	Yes	Yes	Yes	Yes
Hierarchical Data	Yes (Bottleneck - Joins)	Nested Documents	-	-	Yes	Yes
Integrity Constraints	Yes (Enforced by DBMS)	Document Encoding	-	-	Key	Edge has 2 vertexes
Storage	Tables	XML,JSON	Tables	Tables	Graph	Triple
Languages	SQL	XQUERY	Redis	CQL	GREMLIN, CIPHER	SPARQL
Databases	Oracle	MongoDB , CouchDB, Cassandra	Redis , CouchDB	DynamoDB, OracleNOSQL	Neo4j	GraphDB, OracleNOSQL

NOSQL Data Models :

Description	Key Value Stores	Document Stores
Definition	A **key-value store** is a simple hash table, primarily used when all access to the database is via primary key .	A **document store**, is a computer program designed for storing, retrieving and managing document-oriented information, also known as semi-structured data.
Identifier	**Key-** constructed by defining meaningful names(entity, attributes): Entity Name + ':' + Entity Identifier +':' +Entity Attribute	Key, Metadata,Indexes
Operations	Set, Get and Delete	Creation, Retrieval, Update and Delete
Supported Databases	Redis, Oracle NoSQL	MongoDB, CouchDB

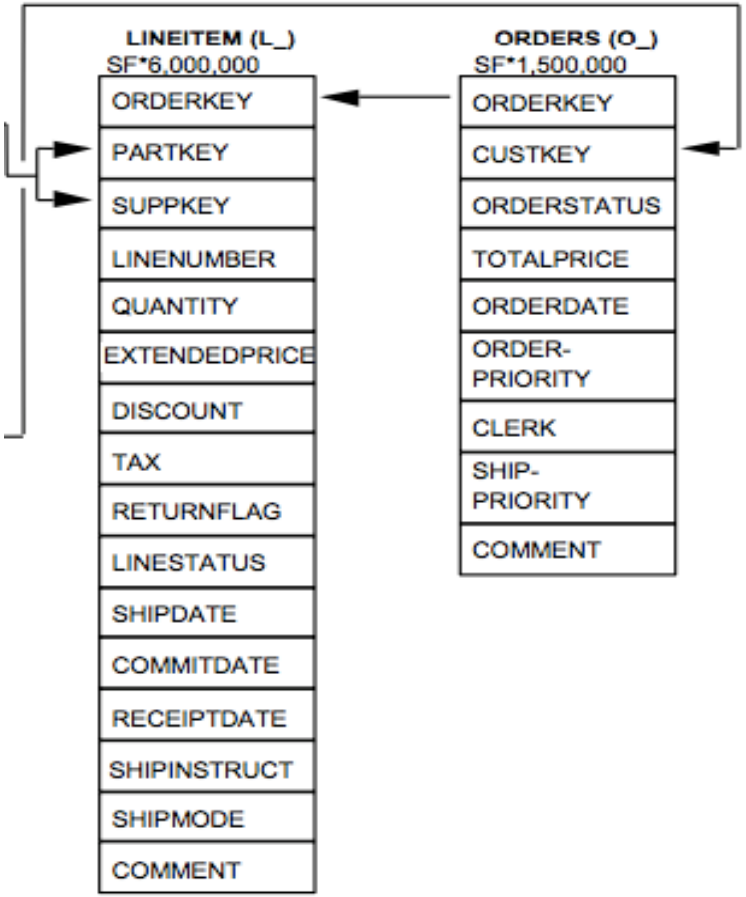


Test Data - TPC-H Datasets

- We are taking two TPC-H datasets (Orders, LineItem).

- Different scale factors of TPC-H datasets provided :

  a) ***Tiny*** – SF 0.1 and SF 0.5
  b) ***Small*** – SF 1 (1.2M Orders and 5.5M LineItem)
  c) ***Medium*** – SF 5
  d) ***Large*** – SF 10
  e) ***Largest*** – SF 15



TPC-H Datasets (Orders, LineItem)

ORDERKEY	CUSTKEY	ORDERSTATUS	TOTALPRICE	ORDERDATE	ORDER-PRIORITY	SHIP-PRIORITY	CLERK	COMMENT
2	7801	O	60951.63	1996-12-01	1-URGENT	Clerk#000000880	0	foxes. pending accounts at the pending silen...
3	12332	F	247296.05	1993-10-14	5-LOW	Clerk#000000955	0	sly final accounts boost. carefully regular id...
4	13678	O	53829.87	1995-10-11	5-LOW	Clerk#000000124	0	sits. slyly regular warthogs cajole. regular ...
5	4450	F	139660.54	1994-07-30	5-LOW	Clerk#000000925	0	quickly. bold deposits sleep slyly. packages u...
6	5563	F	65843.52	1992-02-21	4-NOT SPECIFIED	Clerk#000000058	0	ggle. special final requests are against the ...

ORDERKEY	PARTKEY	SUPPKEY	LINENUMBER	EXTENDEDPRICE	DISCOUNT	TAX	QUANTITY	RETURNFLAG	LINESTATUS	SHIPDATE	COMMITDATE
1	6731	732	2	36	58958.28	0.09	0.06	N	O	1996-04-12	1996-02-28
1	6370	371	3	8	10210.96	0.10	0.02	N	O	1996-01-29	1996-03-05
1	214	465	4	28	31197.88	0.09	0.06	N	O	1996-04-21	1996-03-30
1	2403	160	5	24	31329.60	0.10	0.04	N	O	1996-03-30	1996-03-14
1	1564	67	6	32	46897.92	0.07	0.02	N	O	1996-01-30	1996-02-07
2	10617	138	1	38	58049.18	0.00	0.05	N	O	1997-01-28	1997-01-14

Mapping of RDBMS to Key Value Stores

ORDERKEY	CUSTKEY	ORDERSTATUS	TOTALPRICE	ORDERDATE	ORDER-PRIORITY	SHIP-PRIORITY	CLERK	COMMENT
2	7801	O	60951.63	1996-12-01	1-URGENT	Clerk#000000880	0	foxes. pending accounts at the pending silen...
3	12332	F	247296.05	1993-10-14	5-LOW	Clerk#000000955	0	sly final accounts boost. carefully regular id...
4	13678	O	53829.87	1995-10-11	5-LOW	Clerk#000000124	0	sits. slyly regular warthogs cajole. regular ...
5	4450	F	139660.54	1994-07-30	5-LOW	Clerk#000000925	0	quickly. bold deposits sleep slyly. packages u...
6	5563	F	65843.52	1992-02-21	4-NOT SPECIFIED	Clerk#000000058	0	ggle. special final requests are against the ...

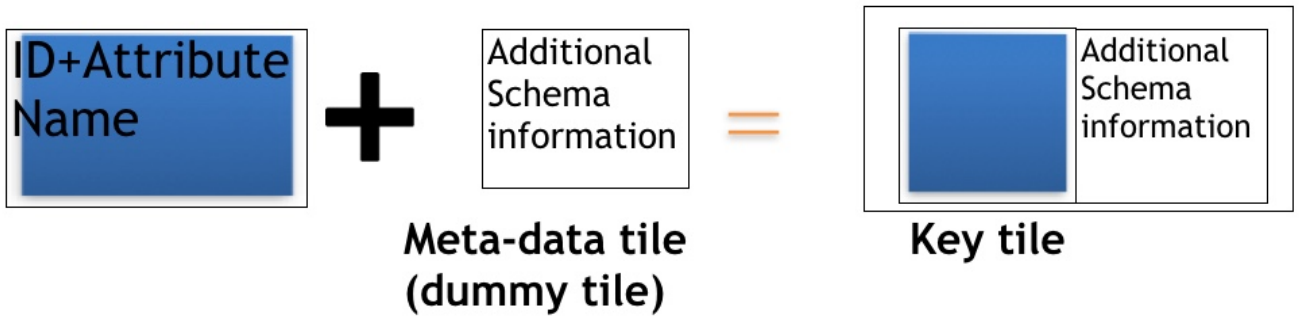
key	Value
2:CUSTKEY	7801
2:ORDERSTATUS	O
2:TOTALPRICE	60951.6
2:ORDERDATE	1996-12-01
2:ORDER-PRIORITY	1-URGENT
2:SHIP-PRIORITY	Clerk#000000880
2:CLERK	0
2:COMMENT	foxes. pending accounts at the pending silen...
3:CUSTKEY	12332
3:ORDERSTATUS	F

- ***How to Construct a Key ?***
  Key = Order Key : Attribute Name
  Value = Value

Implementation of Tiles for KVS

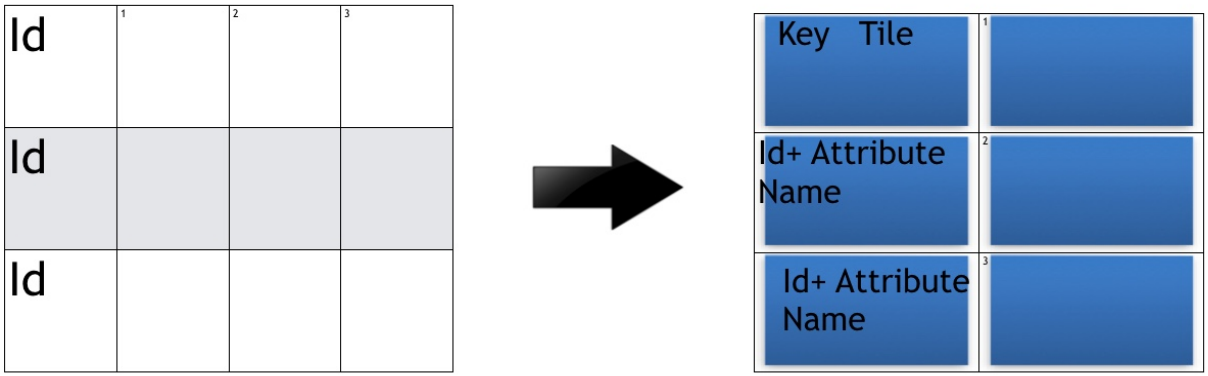
- ***Tile Groups*** - horizontal partitions in Peloton.
- For our mapping to KVS, each row becomes one partition.
- Then, every tile group can be partitioned vertically(partitioned column by column).

- ***Key Tiles*** - consists of Keys stored in the form of ID+Attribute Name(and additional schema information if applicable).
- We replicate the Id + Attribute names(optionally the additional schema information if present) in the key tiles.

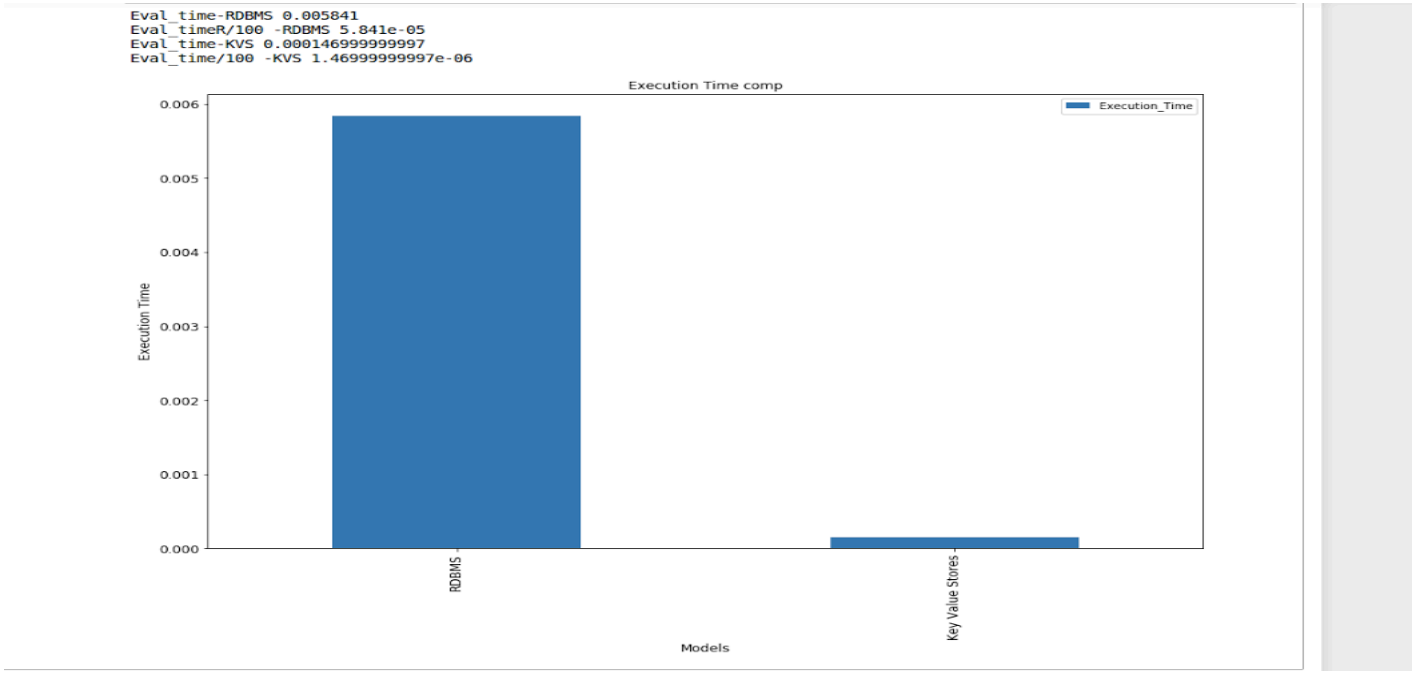


Implementation of Tiles for KVS

- In a rare case if the Id is modified, all the related key tiles need to updated.
- Our KVS proposal acts like a materialized view in the shape of a hash table which maps ID + Attribute name into the value.
- The final KVS representation is depicted in the below figure.

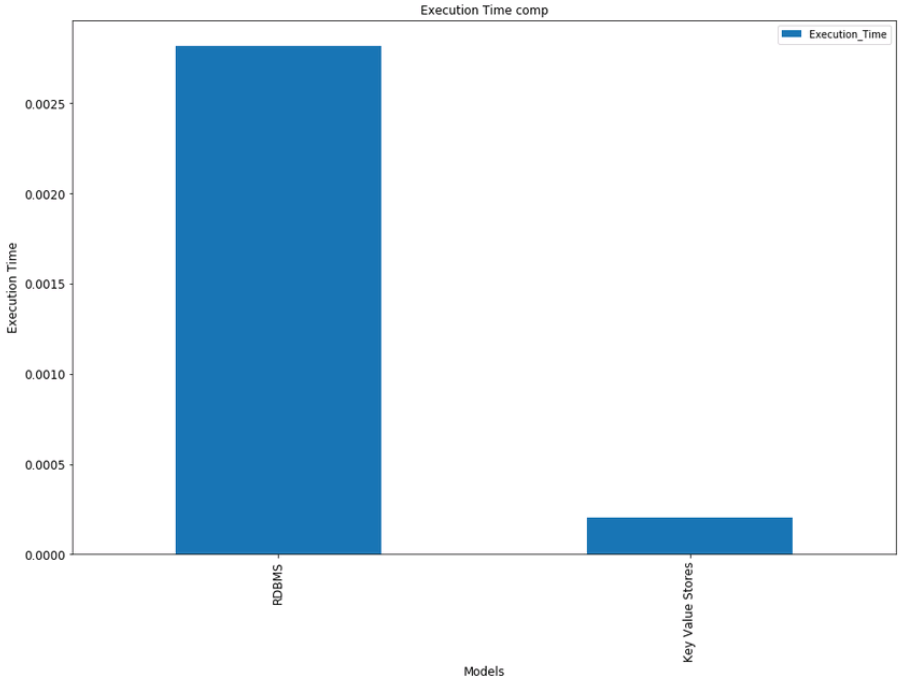


Evaluation Results- RDBMS v/s KVS (Get Operator) SF 0.1

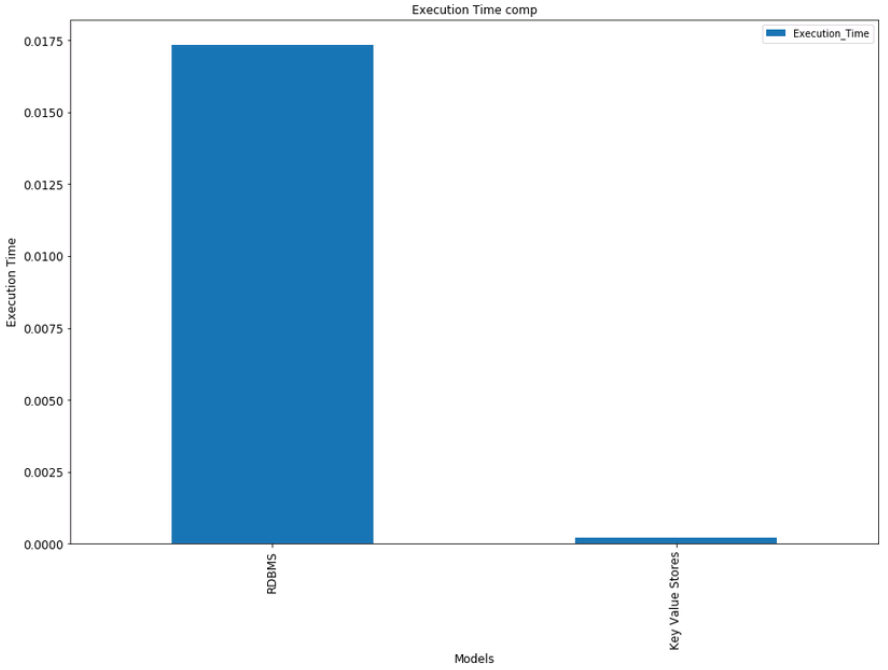


- We compare an unindexed row store against a hash table for point queries.
- A table stored without an index is considered bad for point queries.
- We expect a KVS to be more efficient at retrieving items.
- Better cache use.

KVS - SF 1



KVS - SF 10



Mapping of RDBMS to Document Stores

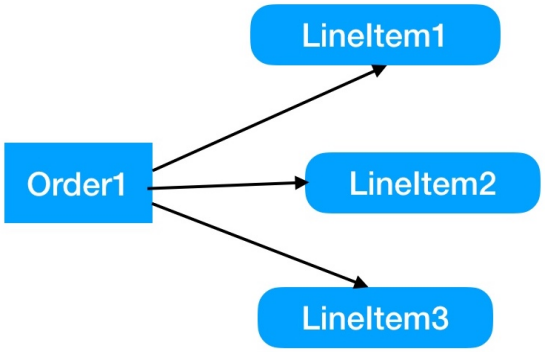
ORDERKEY	PARTKEY	SUPPKEY	LINENUMBER	EXTENDEDPRICE	DISCOUNT	TAX	QUANTITY	RETURNFLAG	LINESTATUS	SHIPDATE	COMMITDATE
1	6731	732	2	36	58958.28	0.09	0.06	N	O	1996-04-12	1996-02-28
1	6370	371	3	8	10210.96	0.10	0.02	N	O	1996-01-29	1996-03-05
1	214	465	4	28	31197.88	0.09	0.06	N	O	1996-04-21	1996-03-30
1	2403	160	5	24	31329.60	0.10	0.04	N	O	1996-03-30	1996-03-14
1	1564	67	6	32	46897.92	0.07	0.02	N	O	1996-01-30	1996-02-07
2	10617	138	1	38	58049.18	0.00	0.05	N	O	1997-01-28	1997-01-14

ORDERKEY	LINENUMBER
1	[2, 3, 4, 5, 6]
2	[1]
3	[1, 2, 3, 4, 5, 6]
4	[1]
5	[1, 2, 3]
6	[1]
7	[1, 2, 3, 4, 5, 6, 7]
32	[1, 2, 3, 4, 5, 6]
33	[1, 2, 3, 4]
34	[1, 2, 3]

- Each Order Item has several LineItems, hence each Order document can have nested LineItem documents.

Implementation of Tiles for Document Stores

- We convert the two tables that are linked by a foreign key, unlike in the case of key value stores.
- To illustrate this mapping, we consider an example wherein an order with Order Id :1 is linked to 3 LineItems as depicted below :-



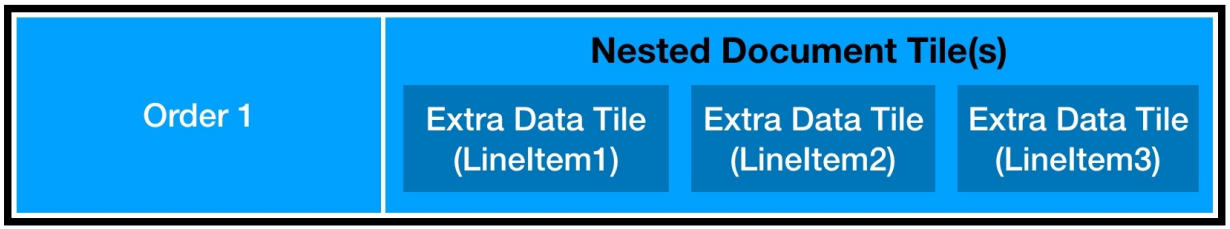
- We do a row per partition but apply no vertical partitions on the base table.
- Corresponding to our example, we only apply partitions on each tuple of the order table (base table) and no partition is applied for the order table attributes.

Implementation of Tiles for Document Stores

- ***Handling Nested Documents*** - We store the foreign key for the LineItem along with the nested document tile(s) as represented below:

- In our next step,

1. We replace Foreign Key with Order Id(Order table).
2. We store each nested LineItem as an ***Extra Data Tile***.



- Denormalised data - as LineItem is stored for each order (precomputed join).

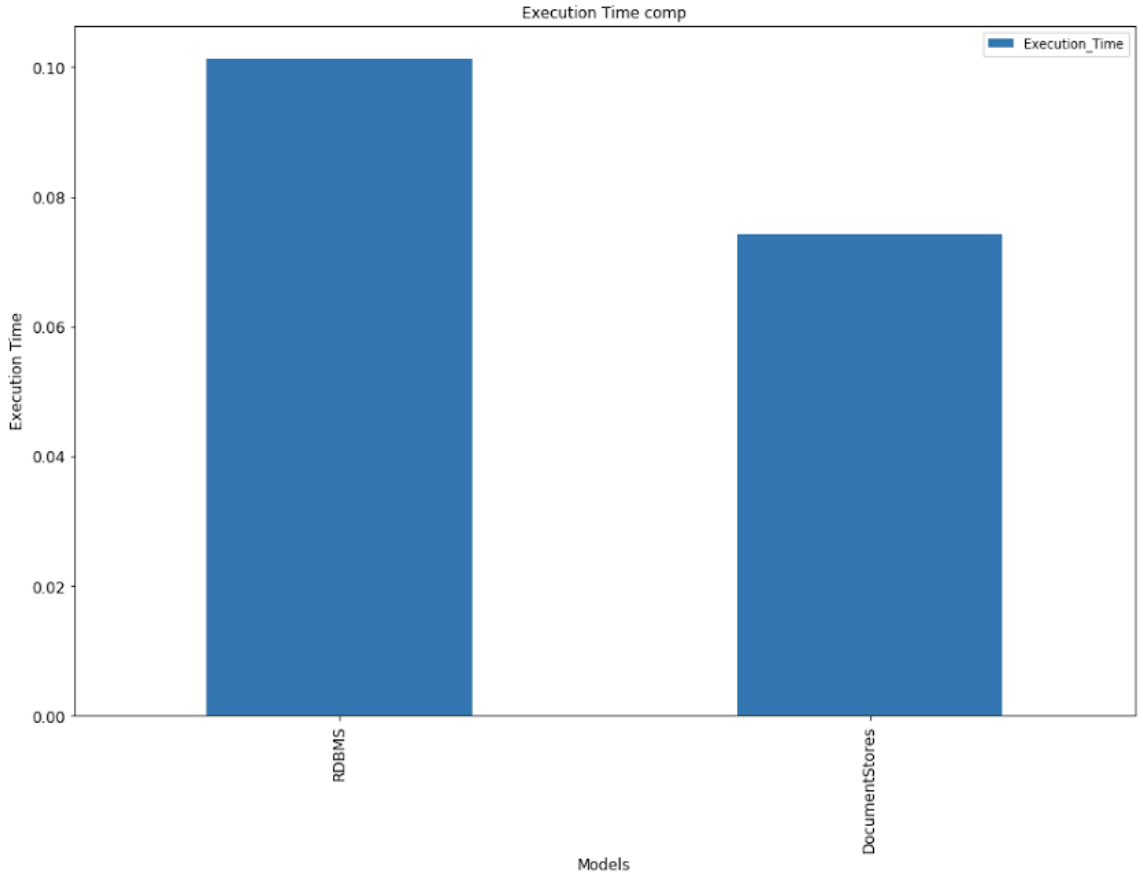
Implementation of Tiles for Document Stores

- This approach of storing the nested document ID's within the base document can be classified as ***CoSQL approach***.
- ***CoSQL(Co relational SQL)*** - Introduces a foreign key relationship from parent to child in addition to the conventional approach that has foreign key relationship from child to parent.
- ***Difference to our mapping*** - CoSQL is one step away from our approach to mapping in which we copy the data itself instead of only the IDs while mapping from document stores to the relational model.

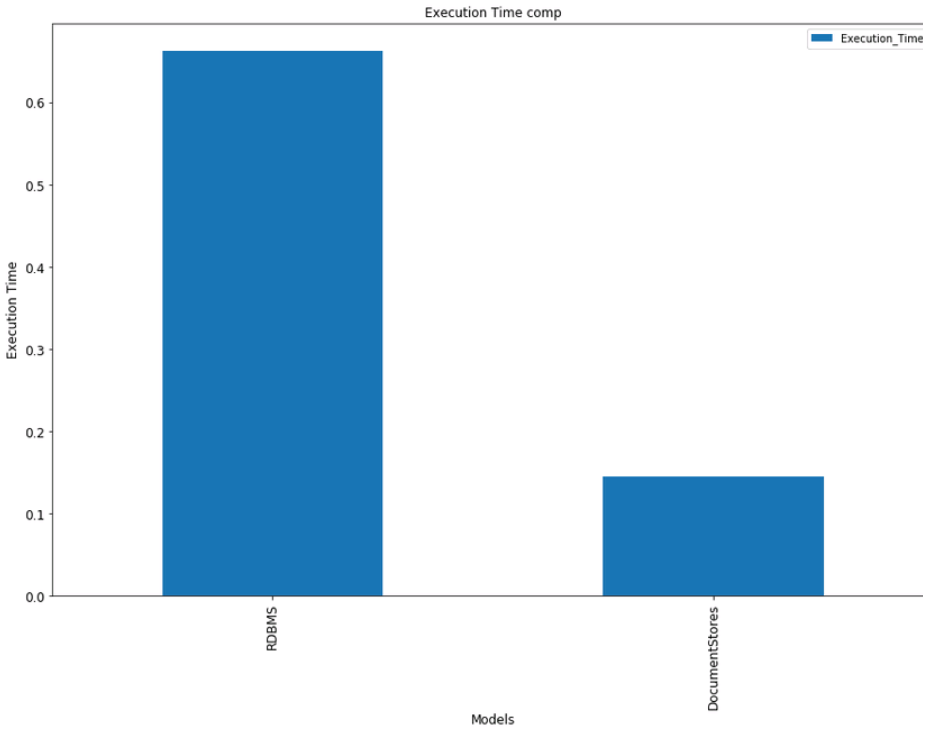
SQL	coSQL
Children point to parents	Parents point to children
Closed world	Open world
Entities have identity (extensional)	Environment determines identity (intensional)
Necessarily strongly typed	Potentially dynamically typed
Synchronous (ACID) updates across multiple rows	Asynchronous (BASE) updates within single values
Environment coordinates changes (transactions)	Entities responsible to react to changes (eventually consistent)
Value-based, strong reference (referentially consistent)	Computation-based, weak reference (expect 404)
Not compositional	Compositional
Query optimizer	Developer/pattern

Evaluation Results- RDBMS v/s DS (Count Operator) SF 0.5

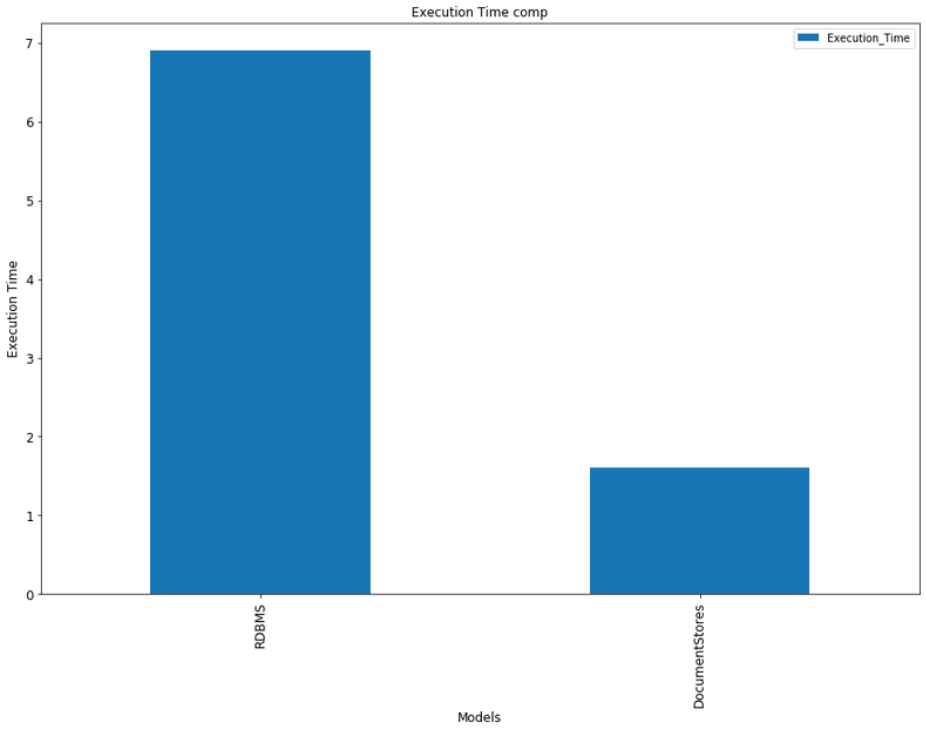
```
Eval_time-RDBMS 0.101311
Eval_timeR/100 -RDBMS 0.00101311
Eval_timeDS 0.0742219999999
Eval_timeDS/100 0.000742219999999
```



DS - SF1



DS - SF10



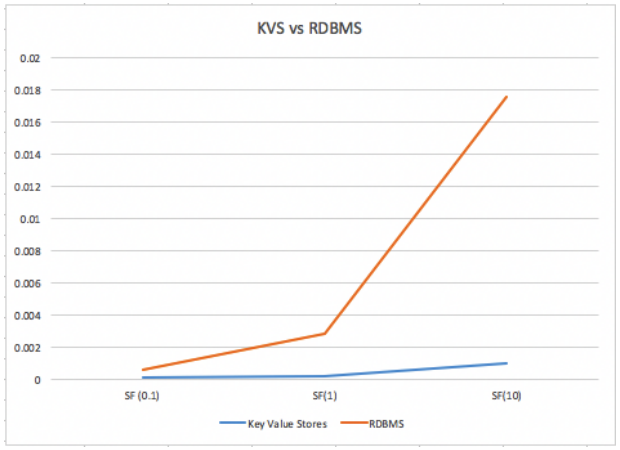
Evaluation Results- RDBMS v/s DS (Count Operator)

- Document store is like a pre computed join. Furthermore, it can have the nested documents with pre computed counts.
- We evaluate a count done by joining and iterating versus a count provided by Python list structure.
- Although not a fair comparison, it would represent the difference between document store and an unoptimised relational model.
- We can also think of it as using a document model, as an index to speed up retrieval and analysis.

Conclusion

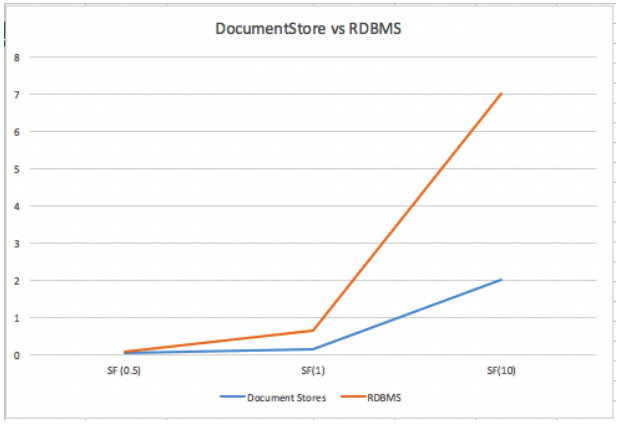
- We proved that there is potential in some non-relational models for queries.
- ***Performance of KVS vs RDBMS***-

1. Sub linear scalability
2. Nearly 4x speed up
3. Predictable and insensitive to the amount of data(bounded)



- ***Performance of DS vs RDBMS*** -

1. DS performance was better compared to relational models.
2. Memory footprint ↑ data size ↑
3. A sharp rise from SF1 to SF10.



Scope for Future Work

- *Generic Framework* - enables the conversion to different models based on tiles.
- *Mapping to Tiles* - other NOSQL models like KCVS, Graph stores etc.
- In future, *extending the Peloton query engine* internally or externally (like SAP HANA) to execute queries that use non relational views.
- *Indexing over views*.
- *Compressed Views* - better storage management.
- As a final note, we recommend studies to *explore and understand the non relational views* could offer.

Project Progress

Milestone	Date	Task Description
Milestone 1	**12.04 – 02.05**	DONE – Initial Research, Presentation and Team Organization
Milestone 2	**02.05 – 23.05**	DONE –Structure of the paper, introduction, Research on models and early implementation setup.
Milestone 3	**23.05 – 13.06**	DONE -First draft of the paper, selection of the operators, early results of the implementation
Milestone 4	**13.06 – 04.07**	DONE - FINAL presentation with the results

Thank you!

Questions?

References

- Tahara, Daniel, and Daniel Abadi. "SQL Beyond Structure: Text, Documents and Key-Value Pairs." *Proc. New England Database Day 2014* (2014).
- Holder, Lawrence B., and Diane J. Cook. "Graph-based relational learning: Current and future directions." *ACM SIGKDD Explorations Newsletter* 5.1 (2003): 90-93.
- De Virgilio, Roberto, Antonio Maccioni, and Riccardo Torlone. "Converting relational to graph databases." *First International Workshop on Graph Data Management Experiences and Systems*. ACM, 2013.
- Wiese, Lena. *Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases*. Walter de Gruyter GmbH & Co KG, 2015.
- Meijer, Erik, and Gavin Bierman. ”A co-relational model of data for large shared data banks.” Communications of the ACM 54, no. 4 (2011): 49-58

- https://en.wikipedia.org/wiki/Document-oriented_database
- http://www.tobez.org/presentations/2011-NPW-redis.html
- https://en.wikipedia.org/wiki/XQuery
- http://www.1keydata.com/datawarehousing/data-modeling-levels.html
- https://en.wikipedia.org/wiki/Data_model
- https://cambridge-intelligence.com/keylines/nosql-key-value-column-stores/