Asynchronous View Maintenance for VLSD Databases

Parag Agrawal, Adam Silberstein, Brian F. Cooper, Utkarsh Srivastava, Raghu Ramakrishnan

SIGMOD 2009

Talk by-
Prashant S. Jaiswal
Ketan J. Mav

Motivation

- Companies have begun to develop Very Large Scale Distributed (VLSD) system to cope up with increasing demands e.g. BigTable, PNUTS etc.
- These system usually only support operations on single records identified through primary key or range scans through key.
- No complex queries like group by aggregation, secondary attribute lookup.
- Query expressiveness is sacrificed for scalability and performance.

Traditional Database Systems	VLSD
High Query Expressiveness. (SQL)	Low Query Expressiveness. (Just Primary Key Operations)
Limited Scalability	High Scalability
Limited Availability	High Availability
High Performance	Latency may increase due to distributed nature.
	BigTable, PNUTS, Cassandra.

Introduction

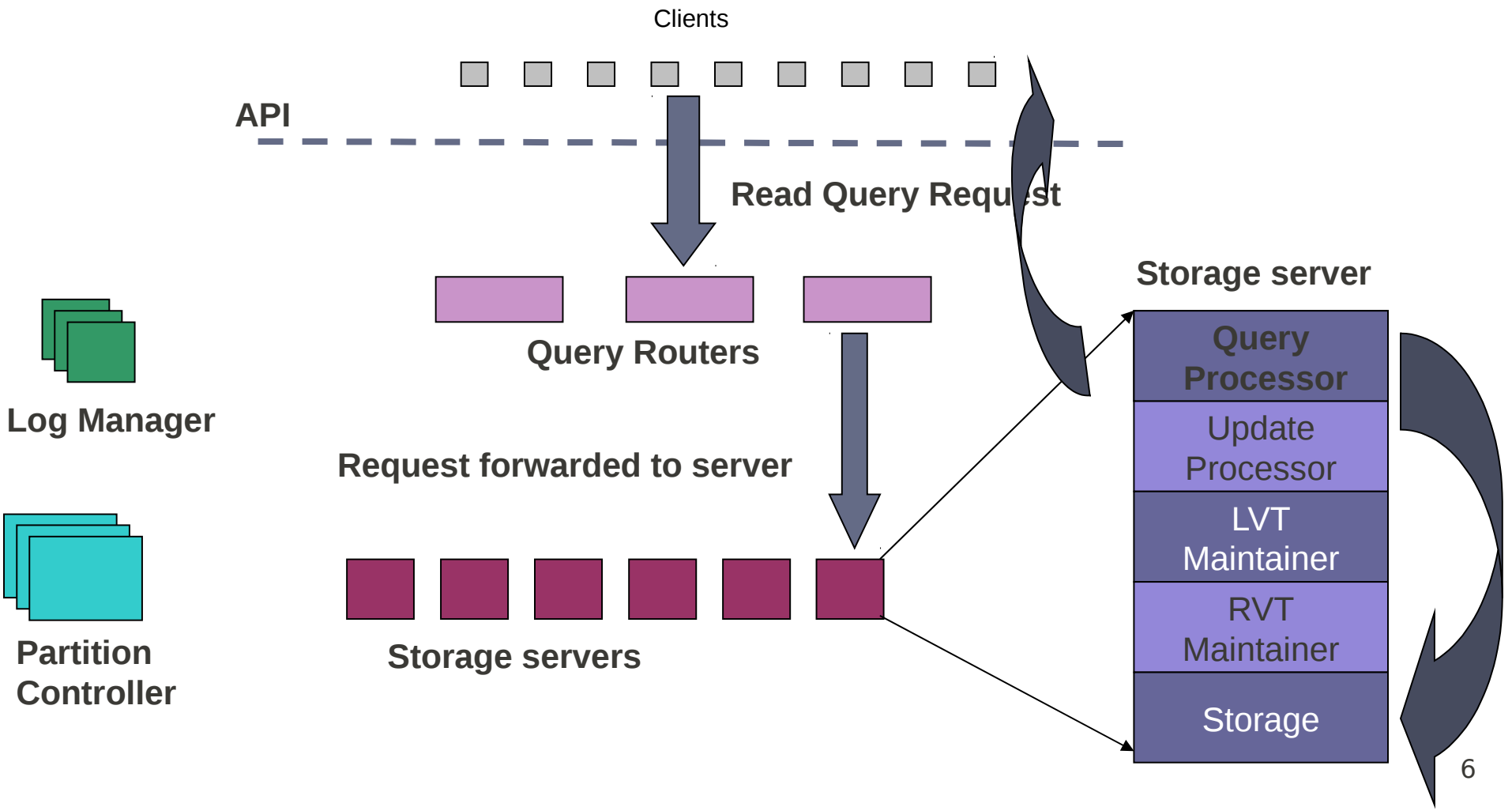
- Complex queries involving joins or aggregates are required in VLSD to improve usability.
- Solution: To create indexes and materialized views.
- Index is just a materialized view with a projection view sorted by one or more secondary attributes.
- Synchronous Updates: Updates are made to the views at the same time as the base table. Query cost increased.
- Asynchronous Updates: Updates are made to the views only after the the base table update is completed. View staleness is an issue here.

Introduction

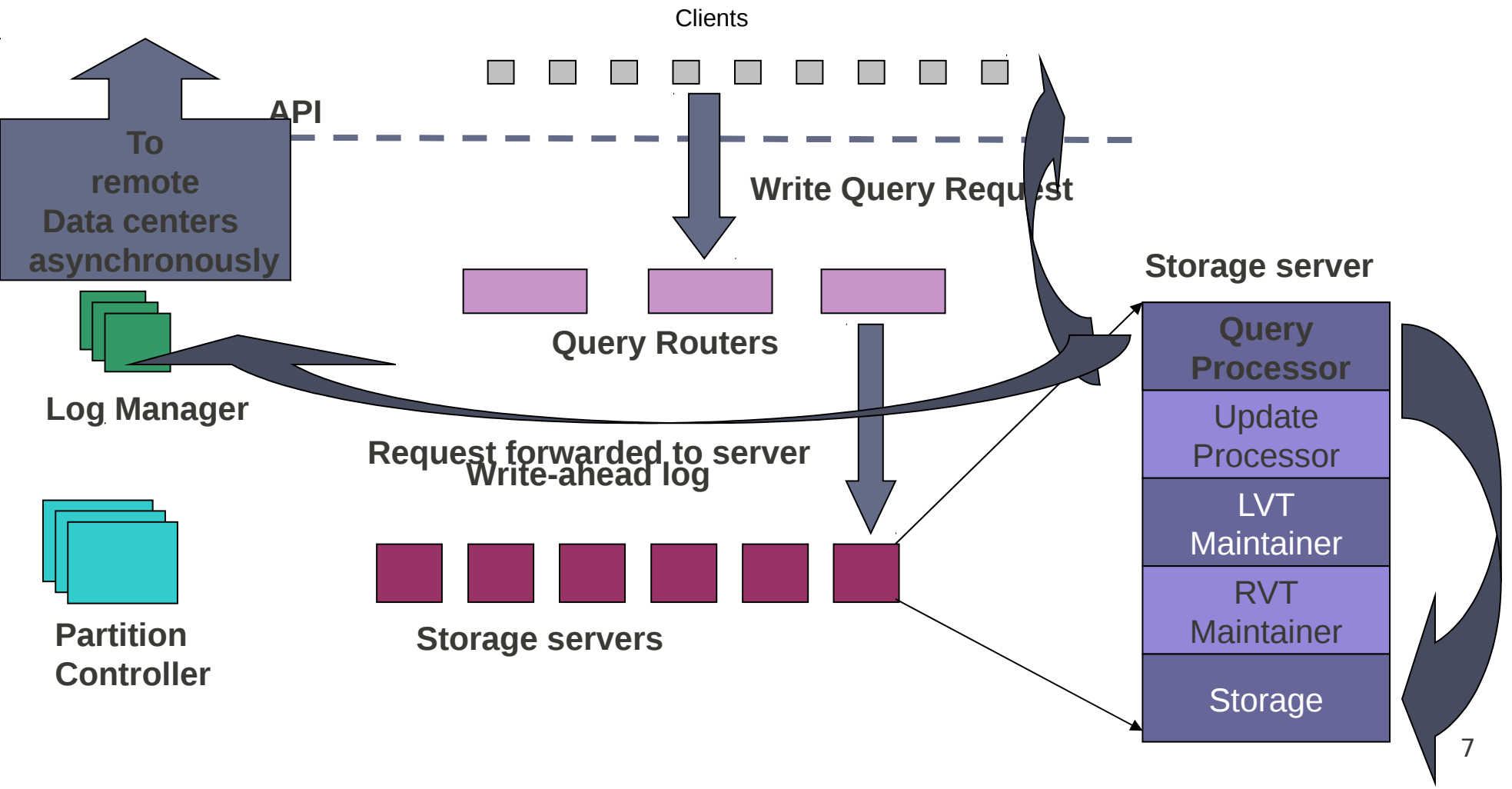
- In VLSD, data is replicated and distributed across servers. So, synchronous updates are expensive in terms of query response time.
- Approach followed: Deferred View Maintenance:

  Update the views only after the base table update completes.
- Decreases query cost and response time improves.
- Challenges are:
  - Ensuring view updates even in failures.
  - Providing consistency guarantees.
  - Replicate the views like base tables.

PNUTS : Architectural Overview



PNUTS : Architectural Overview



Remote View Tables

- RVT is a TABLE separate from the base tables that stores the materialized view over the base table.
- It is maintained asynchronously by RVT maintainer.
- RVTs are partitioned on different key as compared to base table and view records will be on different storage server as compared to base records. Hence called “Remote view tables”.
- Staleness is obvious in RVTs.

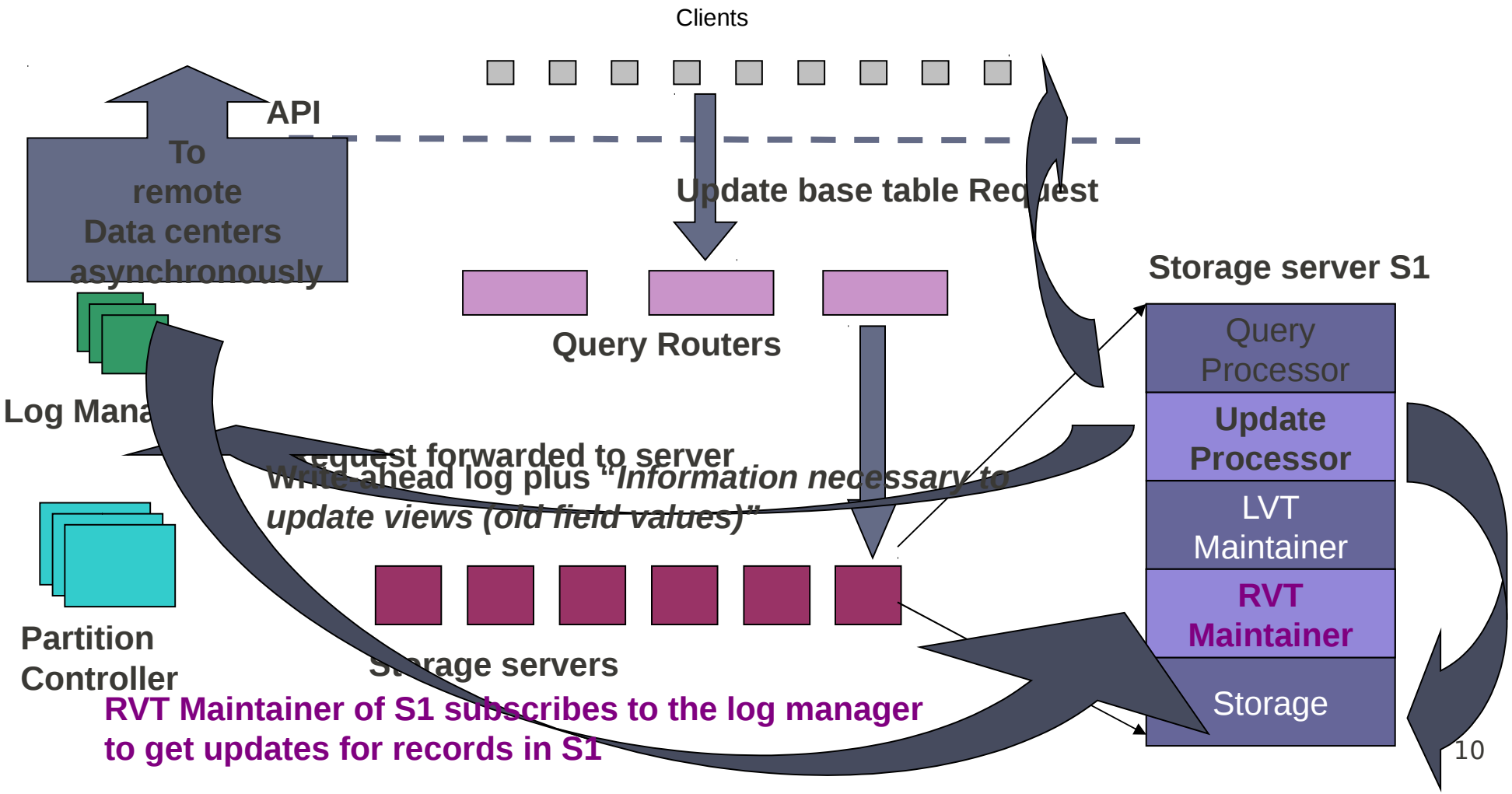
- Base Tables :
  - Items (ItemID, Name, Description , Category, Date, Price, SellerID);
  - Reviews (ReviewID, ItemID, Date, Subject, Rating, Text, Reviewer)
- e.g. CREATE VIEW ByPrice AS

  SELECT Price, ItemID, Name, Category

  FROM Items; (Partitioned on key (Price, ItemID))

Price	ItemID	Name	Category
7000	2	T.V	Electronics
2000	4	Chair	Furniture
2000	1	Car	Toys
50	3	Doll	Toys

Maintaining RVTs



Local View Tables

- Local View Tables
  - Computes on each base partition.
  - Stores the result in the same partition.
  - Partitioned in the same way as base table.
  - Staleness is not the issue.
  - Synchronous update: Updation is the part of same database transaction.
  - Increase in the Query cost.
- e.g.

```
CREATE VIEW ByCategory AS
SELECT Category, COUNT(*)
FROM Items
GROUP BY Category;
```
(Partitioned on ItemID)

LVT Over RVT

- Exploits the fact that RVT is just a normal PNUTS table.
- So, LVT can be built on RVT to collect the partial results of RVTs.
- Generally used to support “Group By” queries.
- Cheaper as the partitioning key of the RVT is same as the group by key of the LVT.
- Data will be pregrouped and aggregates over the RVT are materialized.

Combining RVTs and LVTs

- Main application of LVTs : materialize aggregates over RVTs.
- *E.g. :*

LVT on RVT

Price	Count
7000	1
2000	2
50	1

Price	ItemID	Name	Category
7000	2	T.V	Electronics
2000	4	Chair	Furniture
2000	1	Car	Toys
50	3	Doll	Toys

RVT on Base table

ItemID	Name	Category	Price
1	Car	Toys	2000
2	T.V	Electronics	7000
3	Doll	Toys	50
4	Chair	Furniture	2000
5	Table	Furniture	5000

Base Table

View Types

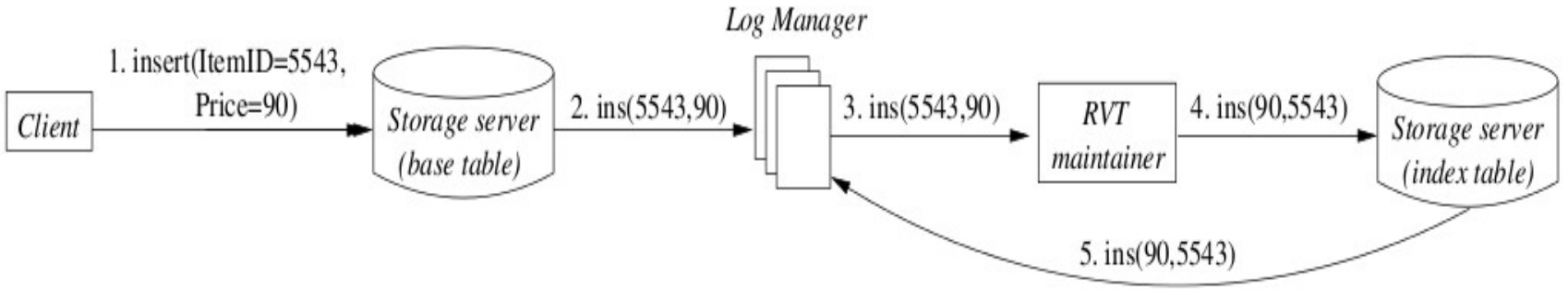
- Limited Query Expressiveness is supported using views like
  - Indexes
  - Equijoins
  - Selections
  - Group By aggregates

View Type: Indexes

- Views are projection with reordering of base table.

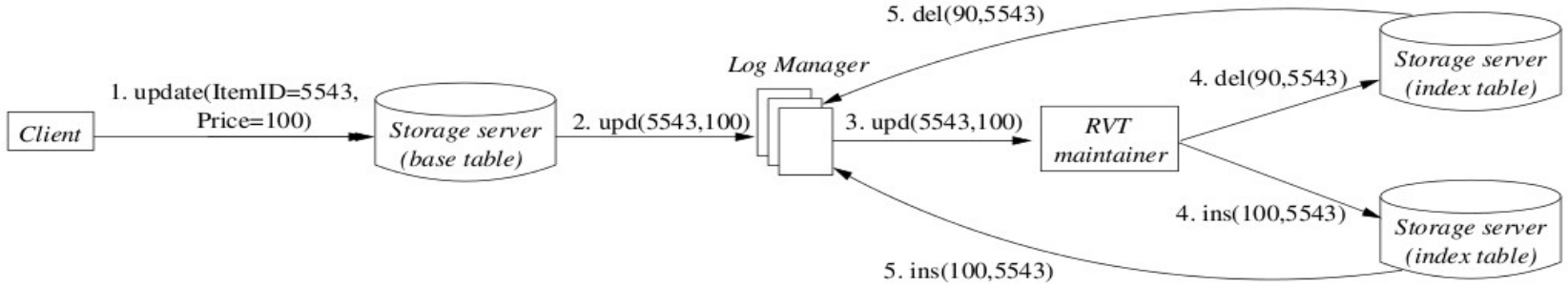
  E.g. RVT : ByPrice(Price, ItemId, Name, Category)

- Name and Category are extra attributes: to satisfy some queries using index only.
- Inserting record is as follows

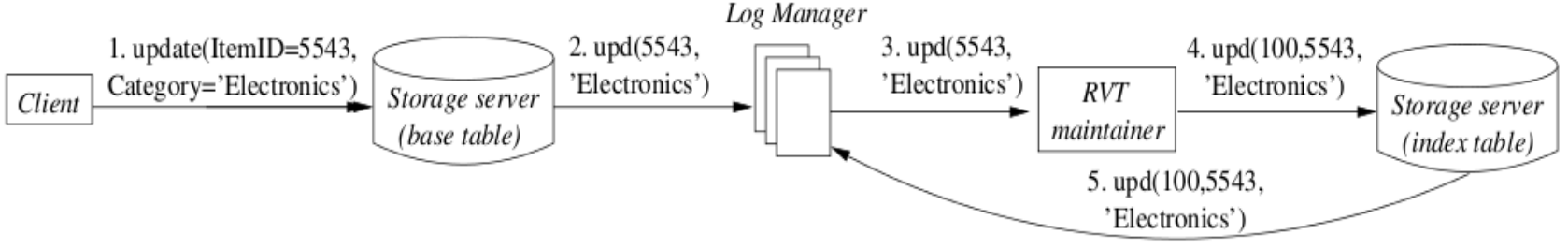


View Type: Indexes

- Updating a key attribute



- Updating a non-key attribute.



View Types: Equijoins

- Co-locating joining records in the same partition but not actually joining them until query time.
- Accomplished using RVT defined as two Indexes on the join-attributes for two relation stored in the same table.
- Outer join is maintained (as we don't materialize join).

ItemId	Name	Category	Price	ReviewerID	Text	Rating
1	Tweety	Toy	1000			
1				123	best	10
2	Car	Toy	2000			
3	Scissor	Stationary -Item	50			
3				234	bad	2

Equijoin

- It increase query time but maintains simplicity and maintenance cost.
- LVT cannot be used as two tables may have different primary keys, Base tables are partitioned differently.
- Equijoin is handled just like Index maintenance (using same mechanism).

View Types: Selections

- Subset of records from the base tables.
- E.g.: CREATE VIEW ELECTRONICITEMS

  SELECT * FROM Items WHERE Category='Electronics'
- Maintained as RVT and like an Index.
- LVTs can be used but cost of querying all the storage server to collect partial view records is high.

Group-by Aggregates

- Multiple base records contribute to one view record, so cant be achieved using RVT index mechanism.
- RVT is replica of single record: cannot be used.
- Generally, LVTs are used to support aggregation either
  - By maintaining LVT over base table
  - Or by maintaining LVT over RVT.
- For min(max) queries, scan one partition for new min(max).
- LVT (over base or over RVT) is maintained synchronously. (Query time increased)

Group-By aggregation views

Base Table

ItemID	Name	Category	Price
1	Car	Toys	2000
2	T.V	Electronics	7000

Partition 1

LVT on Partition 1

Price	Count
7000	1
2000	1

ItemID	Name	Category	Price
3	Doll	Toys	50
4	Chair	Furniture	2000

Partition 2

LVT on Partition 2

Price	Count
2000	1
50	1

Group-By aggregation views

Base Table

ItemID	Name	Category	Price
1	Car	Toys	2000
2	T.V	Electronics	7000

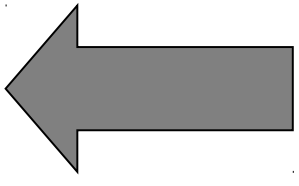
Partition 1

ItemID	Name	Category	Price
3	Doll	Toys	50
4	Chair	Furniture	2000

Partition 2

RVT on Base table

Price	***ItemID***	Name	Category
7000	**2**	T.V	Electronics
2000	**4**	Chair	Furniture
2000	**1**	Car	Toys
50	**3**	Doll	Toys



LVT on RVT

Price	Count
7000	1
2000	2
50	1

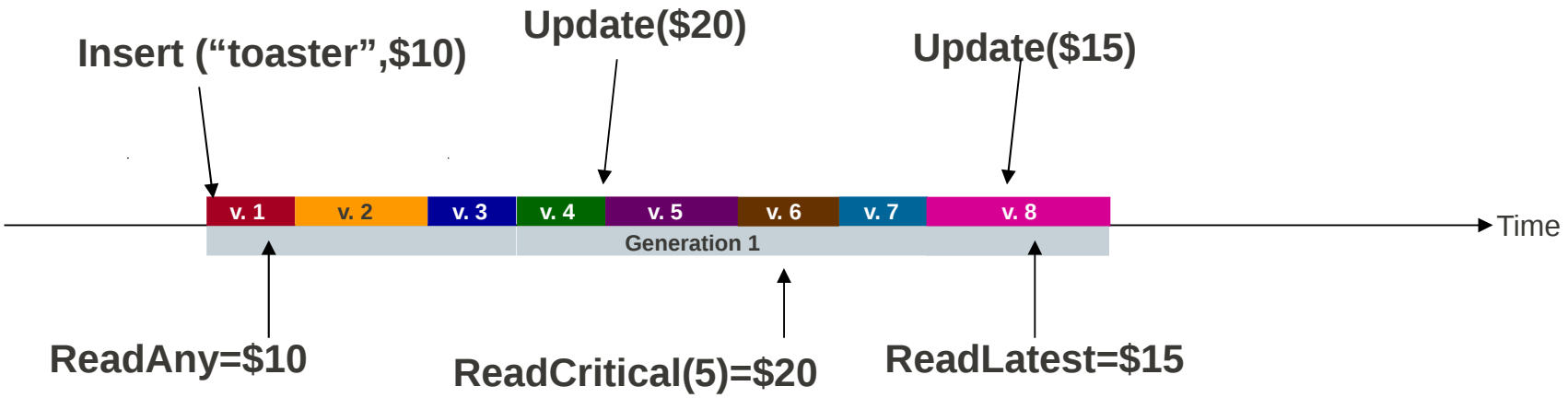
Unsupported View Definitions

- Joins of three or more tables not all joined on same attributes.
  - Joinable records are not located in the same partition
- Joins that are not equijoins.
  - Complex and expensive
- Full SQL99 aggregate functions:

Design Rationale

- View Maintenance: Client or System
  - Not exposing log to the client
  - Avoid Redundant Redo logs maintained by client.
- Eagerness of Updation
  - Synchronous
    - Used for LVT.
  - Lazy Updates
    - Used for RVT.
  - Batched Lazy Updates
    - Not used in order to reduce view staleness.
  - Periodic View Refresh
    - High throughput, staleness but wasted effort

Consistency model



Maintaining View Consistency

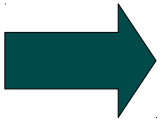
- Base consistency model: timelines of all records are independent
- Multiple records of the views are connected to base records.
- **Information in a view record vr comes from a base record r, vr is dependent on r while r is incident on vr.**
- Indexes, selections, equi-joins: one to one
- Group by aggregates: Many to one

Cost of view maintenance

- Log record of “br” keeps the information needed to update the views.
- For an update to base record br
  - Indexes: If update to key of view, then two updates, else one update.
  - Equi-joins: Same as indexes.
  - Selection views: at most single view update.
  - Group-by aggregate views: for sum/count, view can be updated by knowing only the change in the aggregated field of br and the value of the grouped attribute.

One To One View

ItemID	Name	Category	Price
1	Car	Toys	2000
2	T.V	Electronics	7000
3	Doll	Toys	2000
4	Chair	Furniture	2000



Projection

Price	***ItemID***
2000	**1**
7000	**2**
2000	**3**
2000	**4**

Many To One View

ItemID	Name	Category	Price
1	Car	Toys	2000
2	T.V	Electronics	7000
3	Doll	Toys	2000
4	Chair	Furniture	2000

Group By

Price	Count
7000	1
2000	3

Read Consistency for views:RVT

- Single record reads for one-to-one views:
  - Each view record has single incident base record
  - View records can use the version no. of the base records
  - Consistency guarantees are inherited from the base table reads: ReadAny(vr), ReadCritical(vr, v'), ReadLatest(vr)
- Single record reads for many-to-one views:
  - Multiple br are incident on single vr
  - ReadAny: reads any version of base records.

- Single record reads for many-to-one views (Cont.)
  - ReadCritical: Needs specific versions for certain subsets of base records and ReadAny for all other base records.
  - For e.g. When user updates some tuples and then reads back, he would like to see the latest version of the updated records and relatively stale other records may be acceptable.
  - Base record versions are available in RVT/base table on which view is defined
- ReadLatest: Accesses base table, high cost unavoidable

Read consistency for views: LVT

- LVT is always up-to-date wrt local replica of the underlying base table.
- Thus, any request can be satisfied from the LVT if underlying local base table has correct version.
- ReadAny: may return ver0 if record is absent.
- ReadCritical: In case of staleness, read the relevant records from the base table master.
- ReadLastest: has high cost for RVT because it should access the master base table each time. But cheaper than scanning entire table.
- LVT on RVT: fairly cheap for ReadCritical, expensive for ReadLatest as base table has to be accessed.

Read consistency for views: Range Scans

- Problem of missing and stale records.
- Order of Insertion and Deletion of View records cause extra problems.
  - Insert before delete: record may appear twice
  - Delete before insert: missing record.
- Filter out multiple records corresponding same base record.
- Retain tombstones during deletes.
- Look up base record using key stored in tombstone to include it in scan results.
- Need to garbage collect old tombstones.
- Not implemented in PNUTS yet.

Insert Before Delete

Update price=60 of ItemId=3

Price	ItemID	Name	Category
7000	2	T.V	Electronics
2000	4	Chair	Furniture
2000	1	Car	Toys
50	3	Doll	Toys

Partition 1

Price	ItemID	Name	Category
400	7	T.V	Electronics
200	6	Chair	Furniture
200	5	Car	Toys

Partition 2

Price	ItemID	Name	Category
7000	2	T.V	Electronics
2000	4	Chair	Furniture
2000	1	Car	Toys
50	3	Doll	Toys

Partition 1

Price	ItemID	Name	Category
400	7	T.V	Electronics
200	6	Chair	Furniture
200	5	Car	Toys
60	3	Doll	Toys

Partition 2

Delete Before Insert

Update price=60 of ItemId=3

Price	ItemID	Name	Category
7000	2	T.V	Electronics
2000	4	Chair	Furniture
2000	1	Car	Toys
50	3	Doll	Toys

Partition 1

Price	ItemID	Name	Category
400	7	T.V	Electronics
200	6	Chair	Furniture
200	5	Car	Toys

Partition 2

Price	ItemID	Name	Category
7000	2	T.V	Electronics
2000	4	Chair	Furniture
2000	1	Car	Toys
50	3	Tombstone	

Partition 1

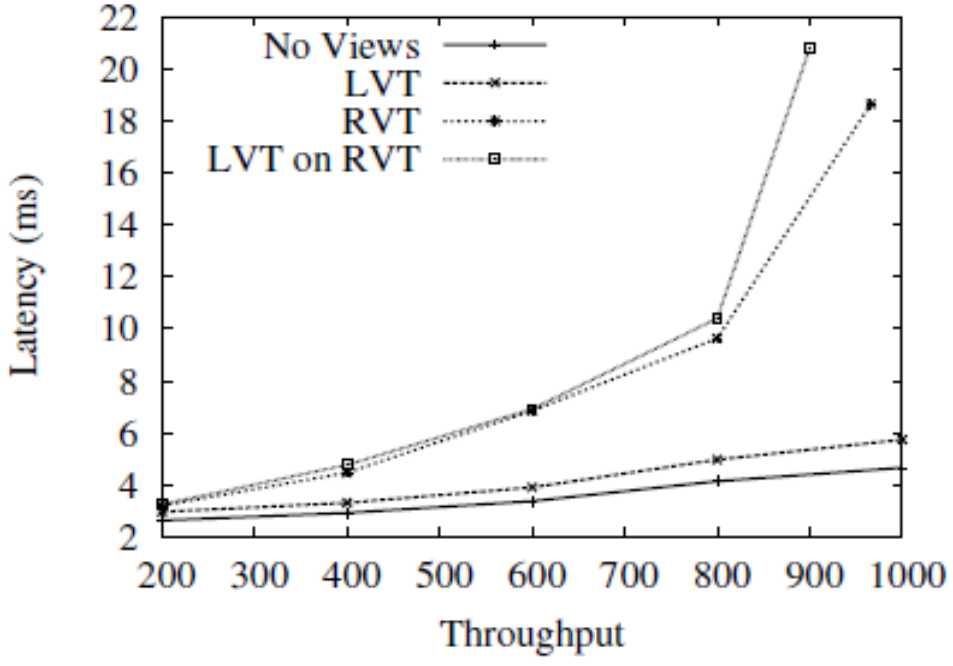
Price	ItemID	Name	Category
400	7	T.V	Electronics
200	6	Chair	Furniture
200	5	Car	Toys

Partition 2

Evaluation

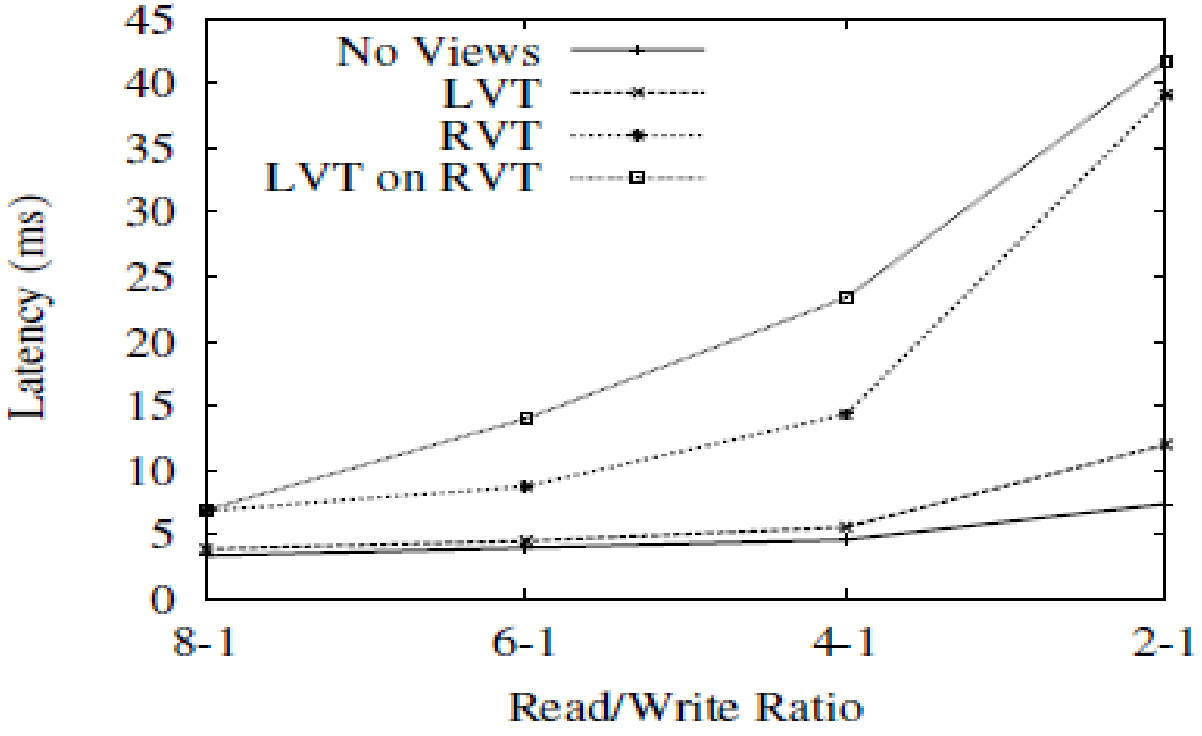
- View maintenance cost is measured upon
  - Latency
  - Throughput
  - Average staleness of the views
- Setup: C++ and Linux/BSD
- Evaluation of costs:
  - 10GB data on each server
  - MySQL buffer pool- 2GB
  - 90% reads served from cache
  - Thin views (indexed attribute and record primary key)
  - I/O bound

Experiment 1: Varying View Type



- Need to provide enough capacity to accommodate extra view maintenance work

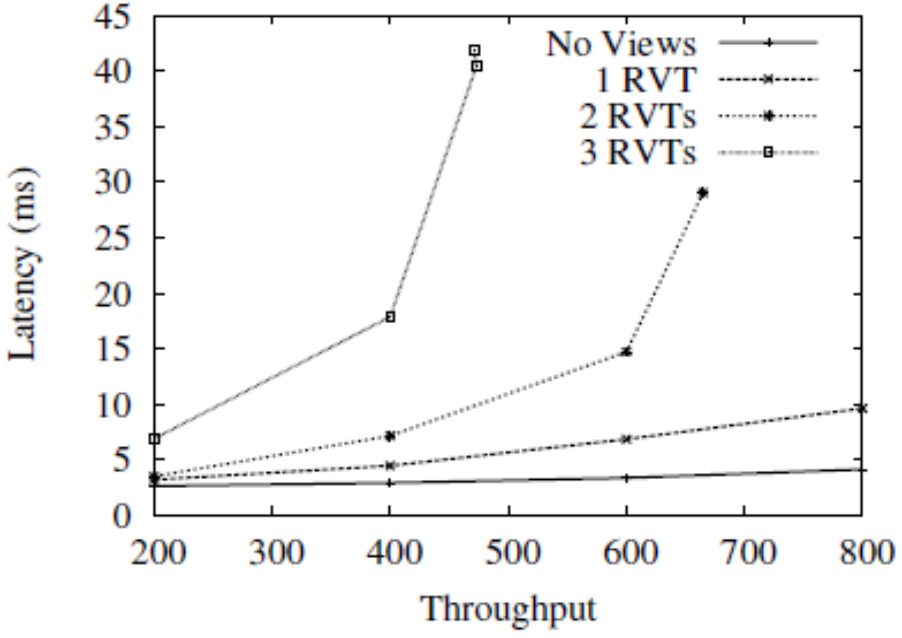
Experiment 2: Varying Read/Write Workload

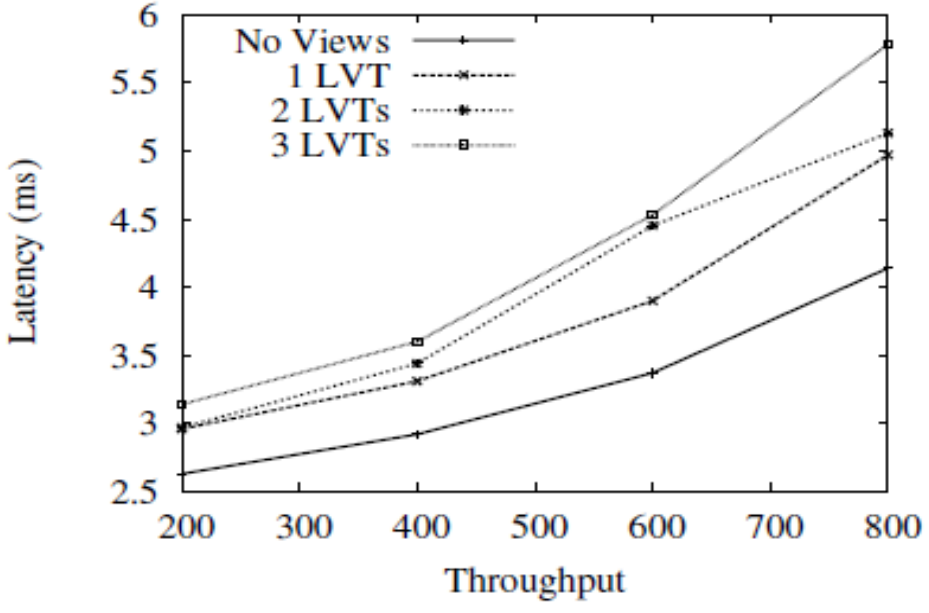


Latency increases with increase in write percentage

Experiment 3: Varying no. of views

- Effect is larger for RVTs than LVTs

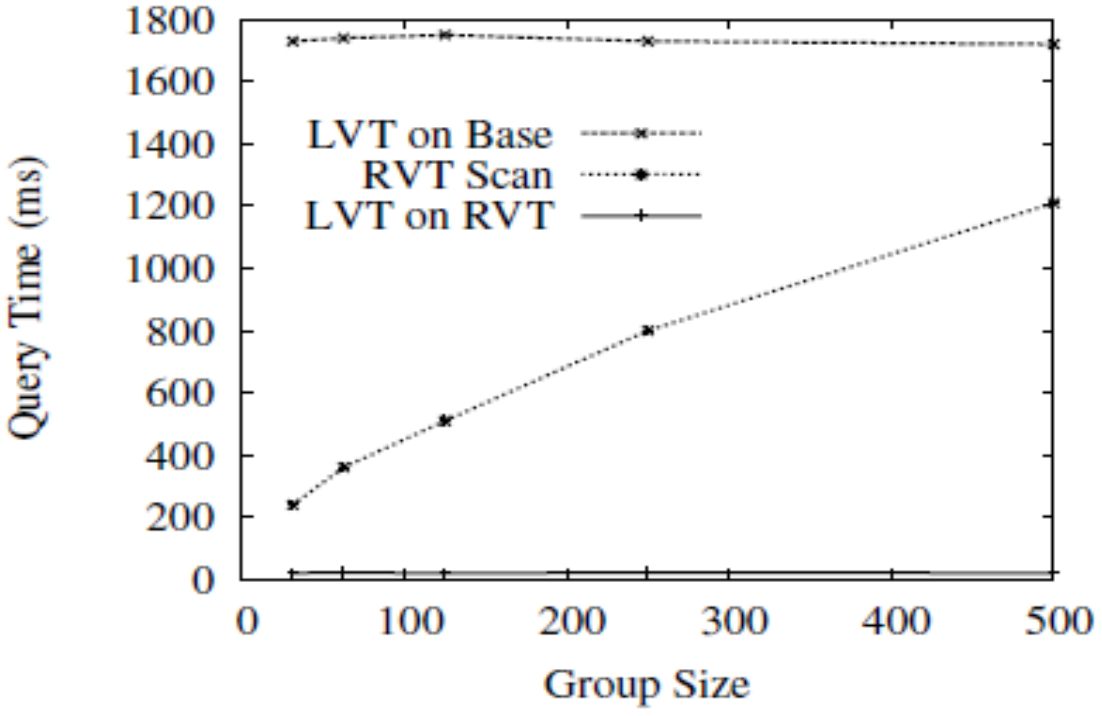




Query Evaluation

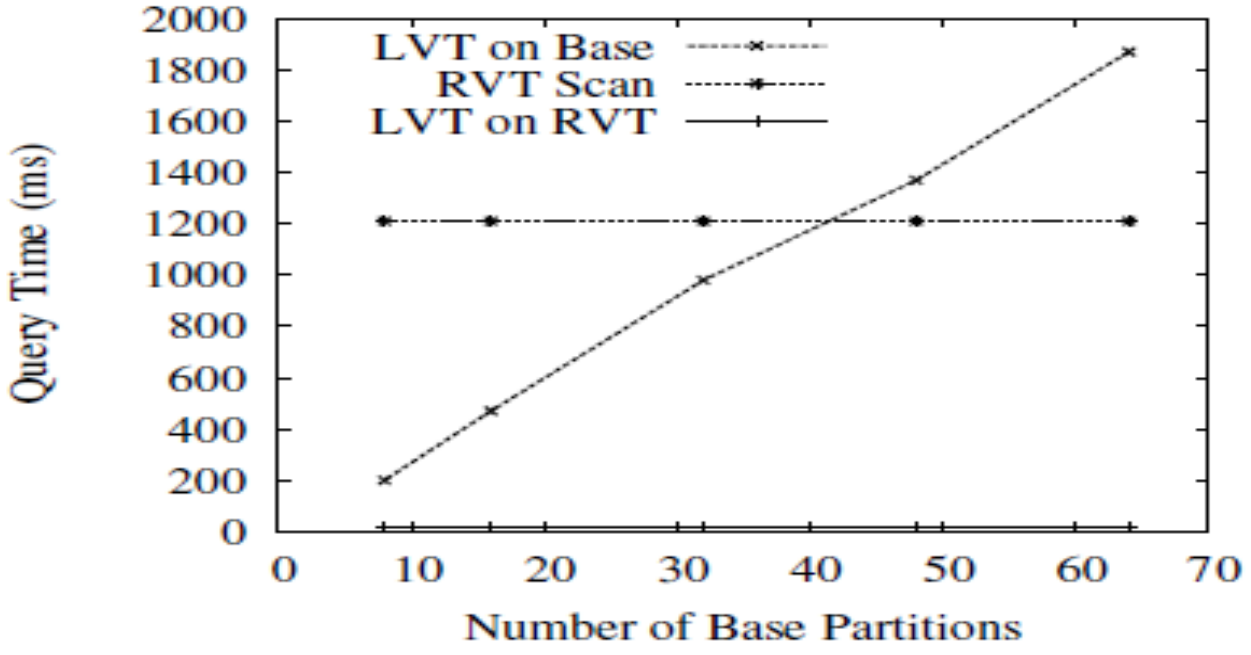
- Index plans
  - Look up on secondary attributes
  - As the resultset size increases, the cost of Index scan also increases.
- Aggregates
  - Index Scan
  - LVT on base
  - LVT on RVT

Query Evaluation : Aggregates



- LVT approaches constant across all group sizes
- LVT-on-base : most expensive
- LVT on RVT : cheapest
- Cost of index scan increases with group size

Query Evaluation : Aggregates



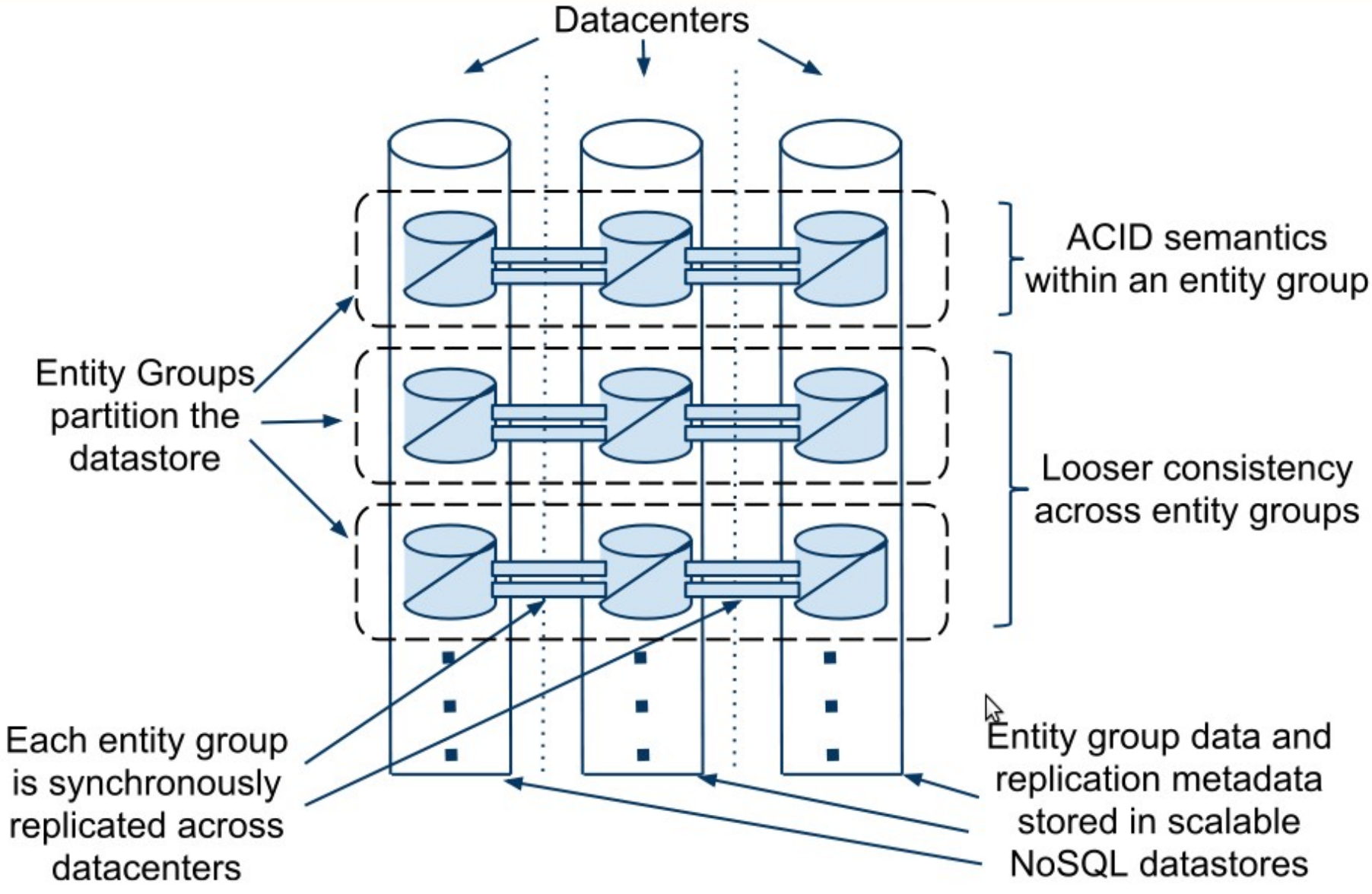
- Fixed group size : 500
- Index scan and LVT on RVT unaffected by no of partitions
- For small partitions, LVT on base beats index scan
- LVT on RVT – best strategy

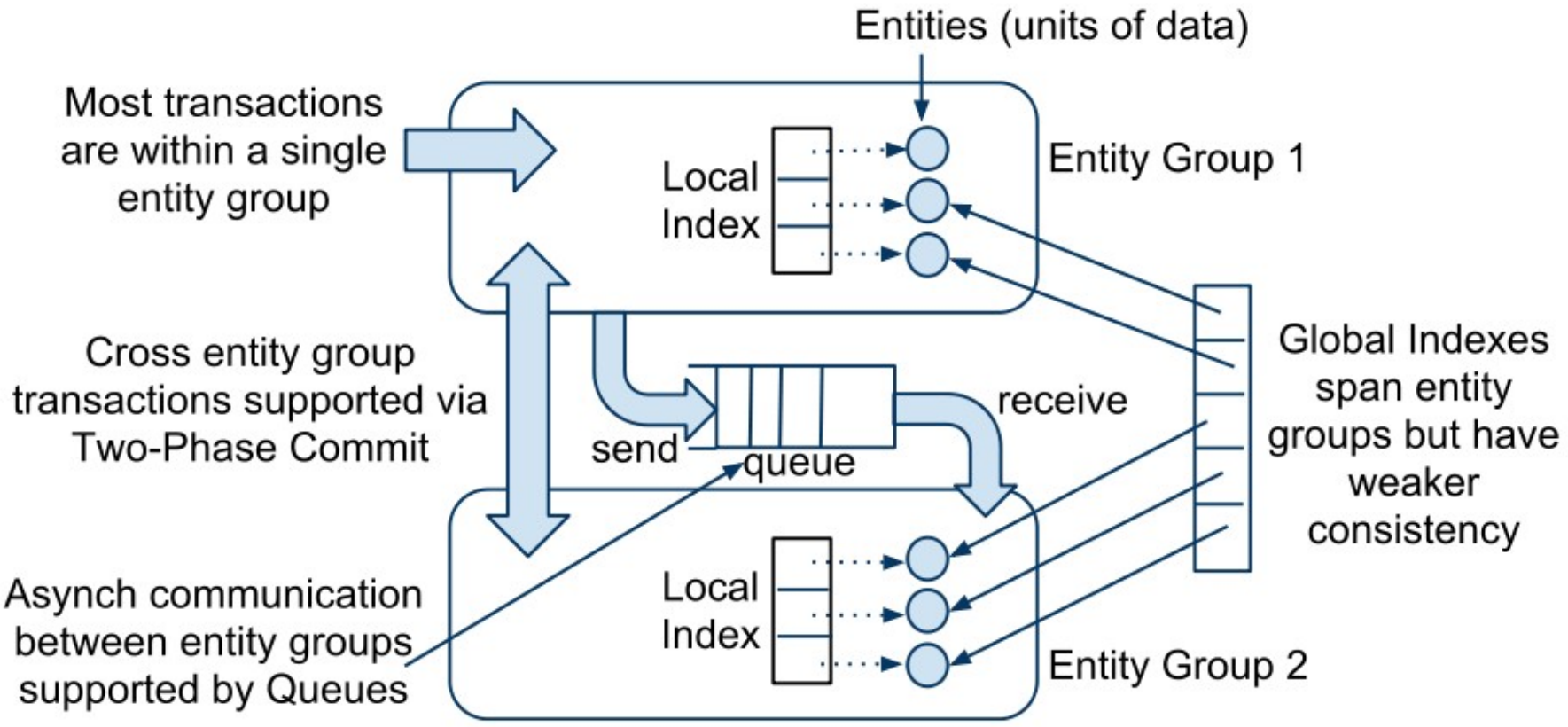
Conclusion

- Maintaining views is essential to enhance the query power in VLSD databases.
- Factors system complexity, throughput and view staleness are to be balanced while selecting the views to be supported.
- RVT: useful for index, equijoin and selection.
- LVT: useful for group-by aggregates.
- Existing PNUTS mechanism for replication and recovery are reused to apply to achieve deferred view maintainance.

Megastore:

- Blends the scalability of NoSQL datastore with the convenience of traditional RDBMS.
- Provides both strong consistency guarantees and high availability.
- Provides fully serializable ACID semantics within fine-grained partitions of the data.
- Synchronous replication of each write across a wide area network with reasonable latency.


Datacenters
ACID semantics within an entity group
Entity Groups partition the datastore
Looser consistency across entity groups
Each entity group is synchronously replicated across datacenters
Entity group data and replication metadata stored in scalable NoSQL datastores


Entities (units of data)
Most transactions are within a single entity group
Local Index
Entity Group 1
Cross entity group transactions supported via Two-Phase Commit
send
queue
receive
Global Indexes span entity groups but have weaker consistency
Asynch communication between entity groups supported by Queues
Local Index
Entity Group 2

Indexes

- Paxos used to deal with Unreachable Replicas.
- Secondary indexes can be declared on any list of entity properties.
- Local index:
  - Treated as separate for each entity group.
  - Is used to find data within an entity group.
  - Stored in the entity group and is updated atomically and consistently with the primary entity data.
- Global index:
  - Spans entity groups
  - Used to find entities without knowing in advance the entity groups that contain them
  - They are not guaranteed to reflect all recent updates.

References

- Asynchronous View maintenance for VLSD databases
  - Parag Agrawal,Adam S, Brian C, Utkarsh S, Raghu Ramakrishnan, SIGMOD 2009
- “PNUTS: Yahoo!’s Hosted Data Serving Platform”
  - Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon,et al
- Megastore: Providing Scalable, Highly Available Storage for Interactive Services
  - Jason Baker et. al. Google, Inc.

Thank You!!