The Consistency Analysis of Secondary Index on Distributed Ordered Tables

Houliang Qi, Xu Chang, Xingwu Liu, Li Zha

University of Chinese Academy of Sciences

中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

Agenda

- Background
- Motivation
- Solutions
  - Consistency
    - Consistency Window
    - Consistency Model
  - Secondary Index
- Evaluation
- Conclusion & Future Work
- Related Work

Background

- NoSQL is widely used
  - The digital universe is huge , and growing exponentially
  - Google BigTable, Yahoo! PNUTS, Apache HBase, Cassandra, etc
  - high performance, low space overhead, and high reliability
- Multi-dimensional Range Queries
  - MDRQ is common
    - latitude > 41.5 and latitude < 44.6 and longitude > 142.5 and longitude < 143.8 and type = 'shop'
  - NoSQL only support row-key query
- Indexing techniques
  - HIndex, Apache Phoenix, Diff-Index etc.

Motivation

- Only concerned about performance, do not care about consistency
- Data inconsistency problem
  - https://issues.apache.org/jira/browse/PHOENIX-3336
- Some questions can’t answer
  - How to measure the consistency between the base table and index table?
  - What is the difference between indexing techniques with eventual consistency models?
  - What can we learn form the above analysis ?

Contribution of our work

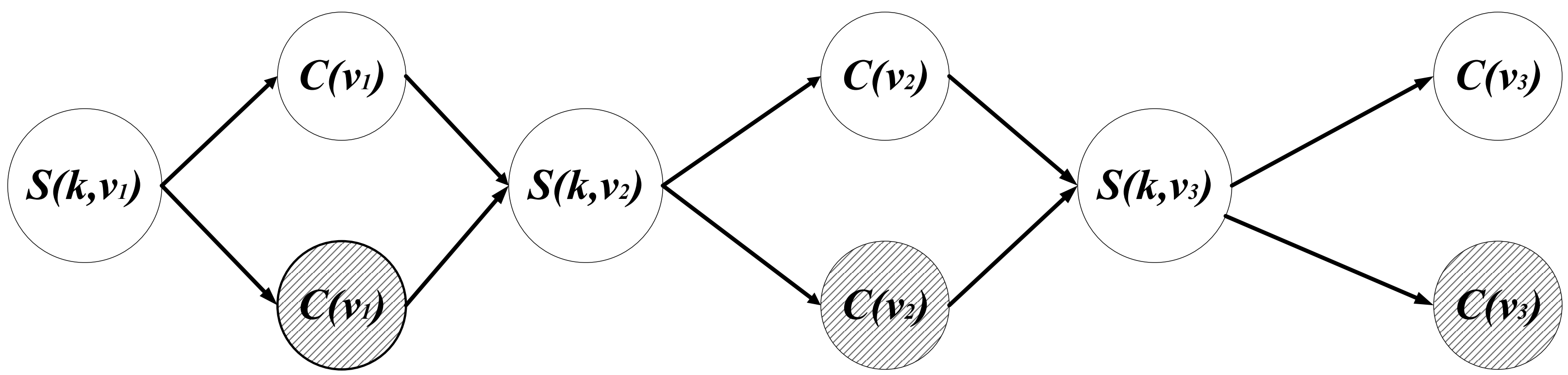
- Introduce the inconsistency window to measure the consistency degree
- Present the definition of strong, RYW and eventual consistency between the index table and base table
- Classify the typical secondary indexing techniques and implement them by ourselves
- Experimental evaluation

Question1

How to measure the consistency between the base table and index table?

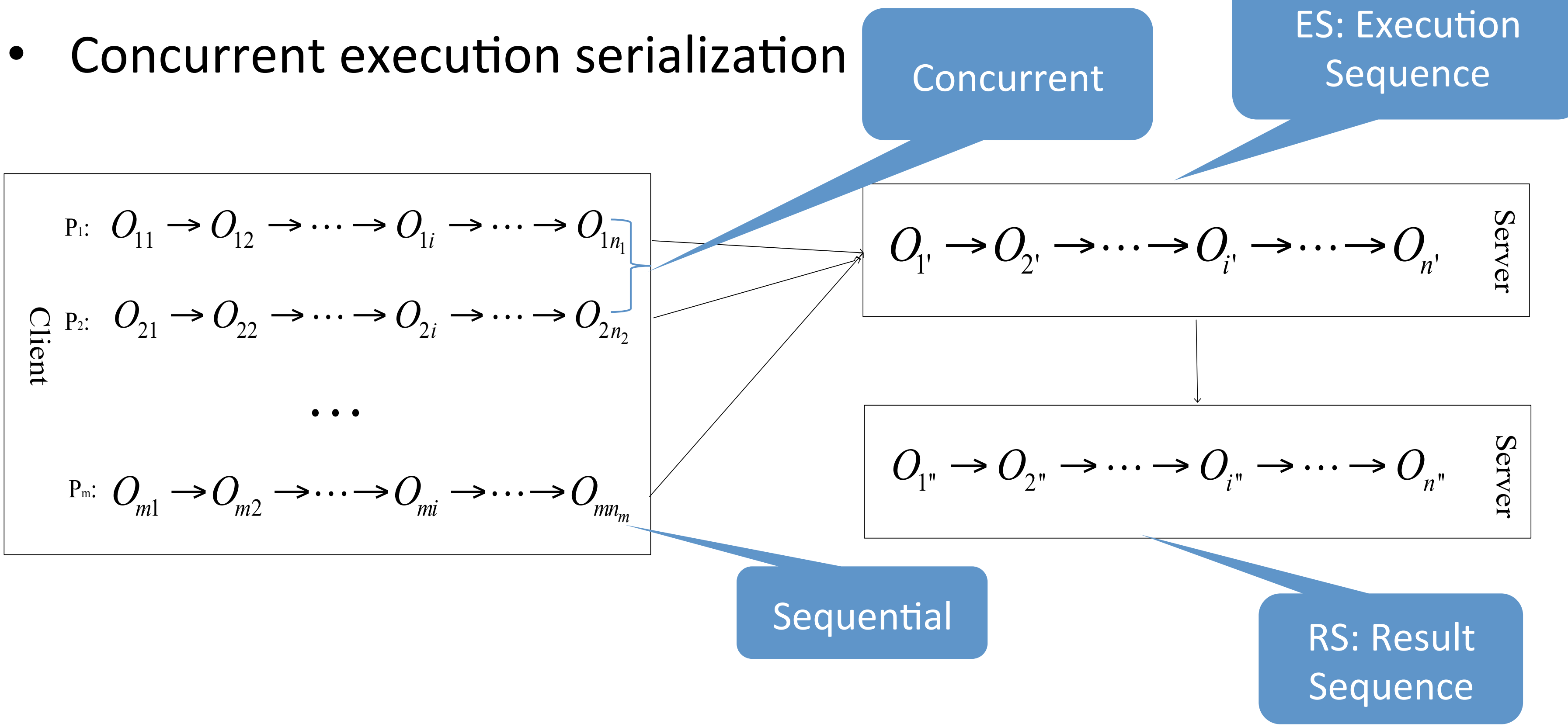
Index model

- Two operations
  - *C(v)*:reading *key* from index table by the value *v* and reading the record < *k, v'* > from base table by the *key*, then comparing *v'* with *v* .
  - *S(k, v)*: writing the record< *k, v* > into the base table and updating the index table
- Execution DAG



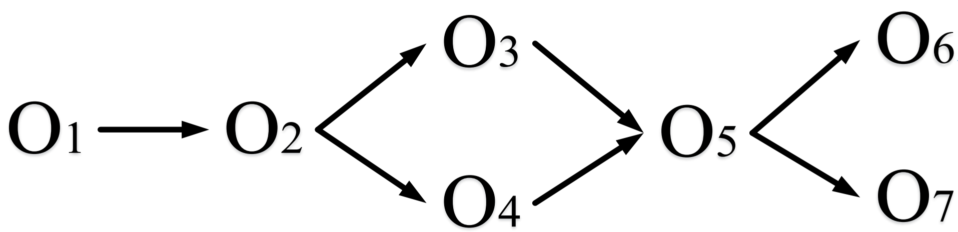
Inconsistency Window

- Concurrent execution serialization



- **Inconsistency Window** :Given any operation *O*, consider the set of operations *O'* preceding *O* in *ES* and after *O* in *RS*, denoted by *Lag(O)*. Define inconsistency window of the *RS* as $max_{O \in RS}$ | *Lag*(*O*)|.

Inconsistency Window Calculate



Execution DAG: from client perspective

Execution Sequence: Sort according to the time at which the operation begin to execute

Result Sequence: Sort by time according to the completion of the operation

Inconsistency Window: For each operation, calculate its distance between the ES and RS

time

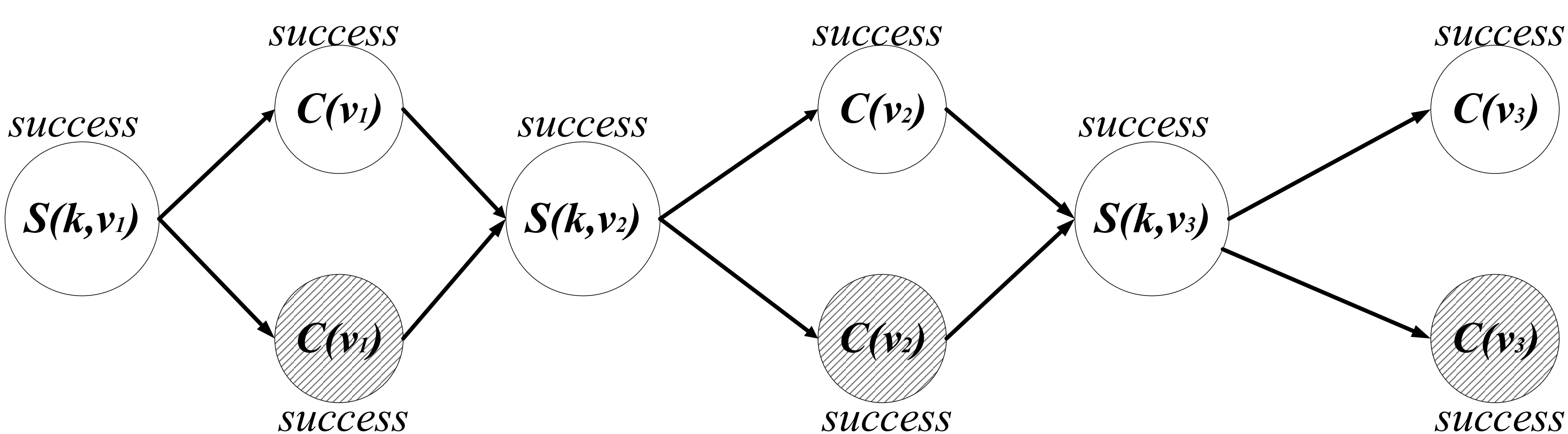
ES
O_1
O_2
O_3
O_4
O_5
O_6
O_7

RS	ICW
O_1	0=|1-1|
O_3	1=|3-2|
O_5	2=|5-3|
O_7	3=|7-4|
O_2	3=|2-5|
O_6	0=|6-6|
O_4	3=|4-7|

Consistency Model: Index table & Base Table

- **Strong Consistency**

  For any execution *ES*, there is a linearization *RS* preserving the order in *ES*, which means that if an operation *O*1 precedes *O*2 in *ES*, then operation *O*1 precedes *O*2 in *RS*.



Consistency Model: Index table & Base Table

- **Strong Consistency**

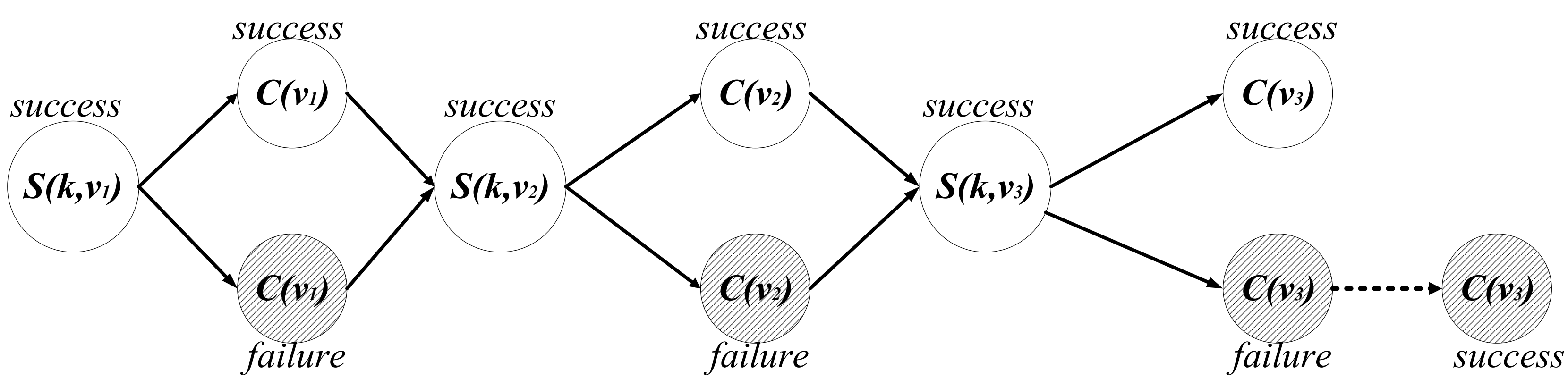
  For any execution *ES*, there is a linearization *RS* preserving the order in *ES*, which means that if an operation *O*1 precedes *O*2 in *ES*, then operation *O*1 precedes *O*2 in *RS*.

- **Read-Your-Writes(RYW) Consistency**.

  There is a number Δ > 0. For any execution *ES*, there is a linearization whose inconsistency window is less than Δ and preserving the ordering of operations launched by any client.

- **Eventual Consistency**.

  There is a number Δ > 0 such that for any execution *ES*, there is a linearization whose inconsistency window is less than Δ



Question 2

What is the difference between indexing techniques with eventual consistency models?

Update operation in DOT

- Distributed Ordered Tables
  - introduced by Yahoo!
  - partitions continuous keys to regions, replicates regions, distributes regions to shared-nothing servers.
  - serves as tables and columns, supports range queries on primary keys.
- Update operation in DOT

Algorithm 1 Update operation in DOT

When an update operation $< k, v_{new}, ts_{new} >$ to base table is observed, do Step1 to Step5

Step1. Put data into base table: $W_B(k, v_{new}, ts_{new})$;
Step2. Put data into index table: $W_I(v_{new}, k, ts_{new})$;
Step3. Read the value of object k before ts_{new}:
$v_{old} \leftarrow R_B(k, ts_{new} - \delta)$
Step4. Delete old index from index table: $D_I(v_{old}, k, ts_{new} - \delta)$
Step5. Delete old data from base table: $D_B(k, v_{old}, ts_{new} - \delta)$

Secondary Indexes

- Indexing techniques classification

No.	Indexing Techniques	Description
1	Sync	Step1+Step2+Step3+Step4
2	Async-insert	Step1+Step2+async maintain
3	Async-compact	Step1+Step2+compact maintain
4	Async-simple	Step1+async maintain

Algorithm 1 Update operation in DOT

When an update operation $< k, v_{new}, ts_{new} >$ to base table is observed, do Step1 to Step5

Step1. Put data into base table:$W_B(k, v_{new}, ts_{new})$;
Step2. Put data into index table:$W_I(v_{new}, k, ts_{new})$;
Step3. Read the value of object k before ts_{new}:
$v_{old} \leftarrow R_B(k, ts_{new} - \delta)$
Step4. Delete old index from index table: $D_I(v_{old}, k, ts_{new} - \delta)$
Step5. Delete old data from base table: $D_B(k, v_{old}, ts_{new} - \delta)$

- Here we only focus on the eventual consistency

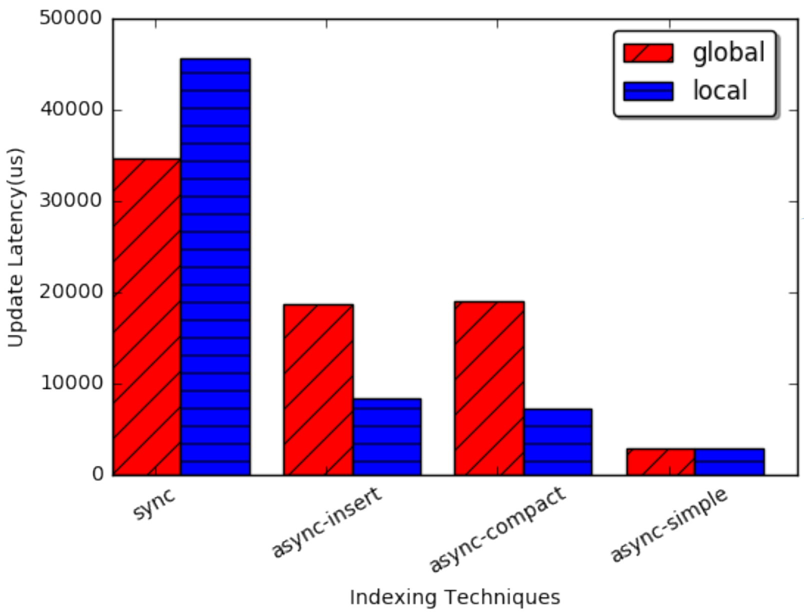
Strong	RYW	Eventual
Google Spanner, XiaoMi Themis, Apache Phoenix	Diff-Index	CMIndex, SLIK , Diff-Index, DELI , HIndex , IHBase , Apache Phoenix

Indexing Techniques	Global	Local
Sync	CMIndex, Diff-Index	Apache Phoenix
Async-insert	Diff-Index, SLIK	-
Async-compact	DELI	HIndex
Async-simple	Diff-Index	IHBase

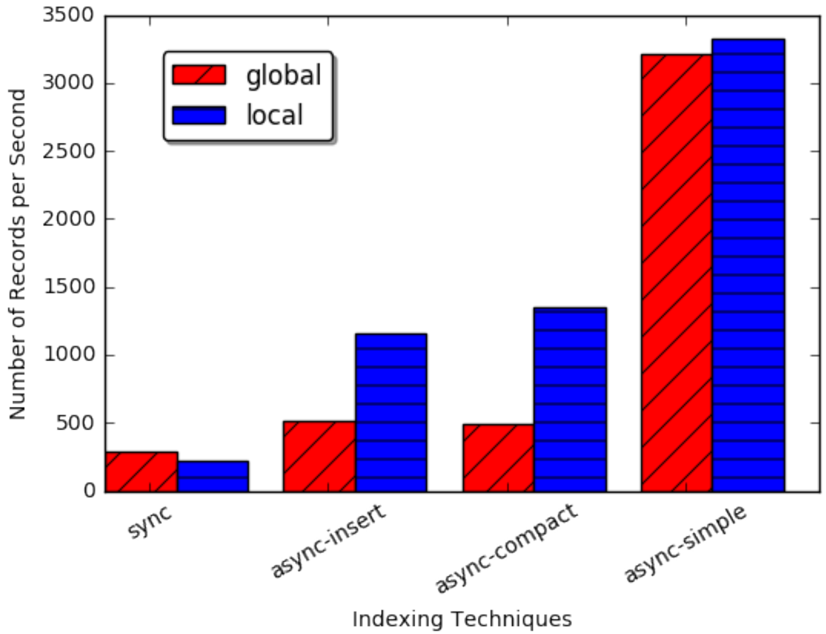
Evaluation

- Experimental setup
  - 7 nodes: 3 client, 1 master, 3 slaves
  - Client: generate workload by YCSB
  - Servers: HBase/HDFS default configuration
- Workload scenario
  - Scenario A: 100% update
  - Scenario B: 50%scan, 50%update

Evaluation- Scenario A



Sync: higher
Async: lower



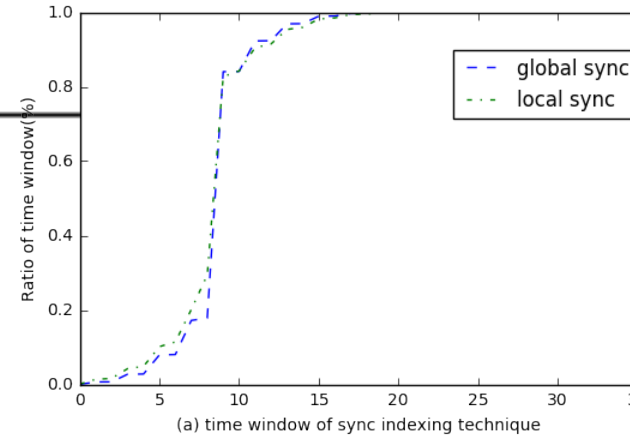
Sync: lower
Async: higher

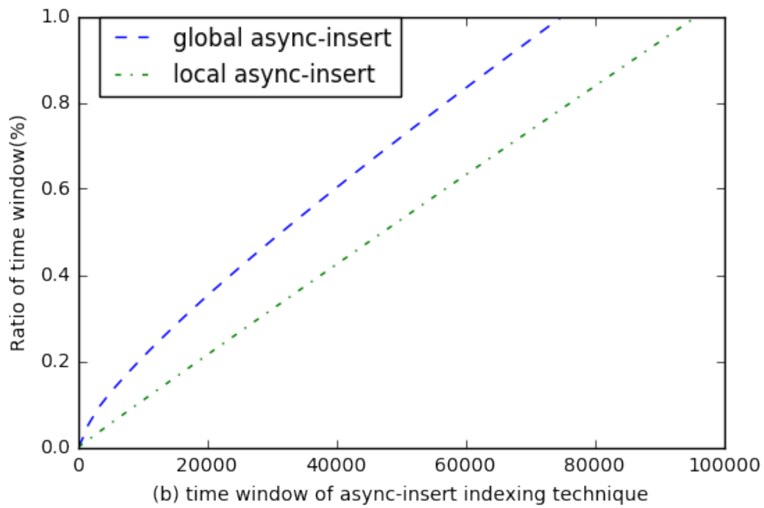
Evaluation- Scenario A

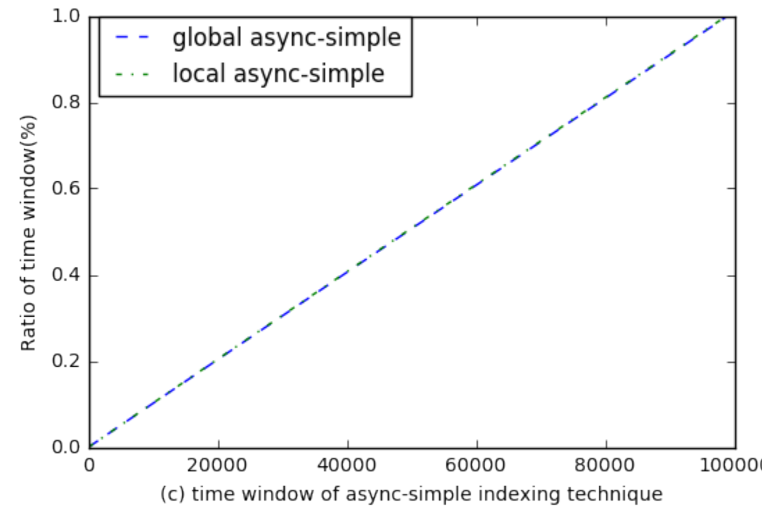
– Global and local have the similar consistency for the same indexing techniques.

– Sync has the highest degree of consistency

– Async-insert and async-simple indexing technique both show worse consistency

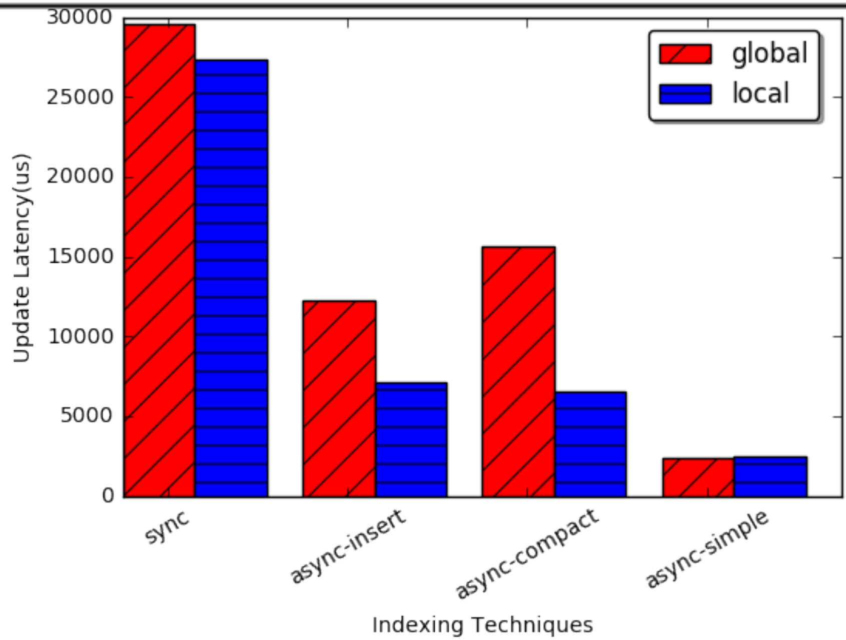


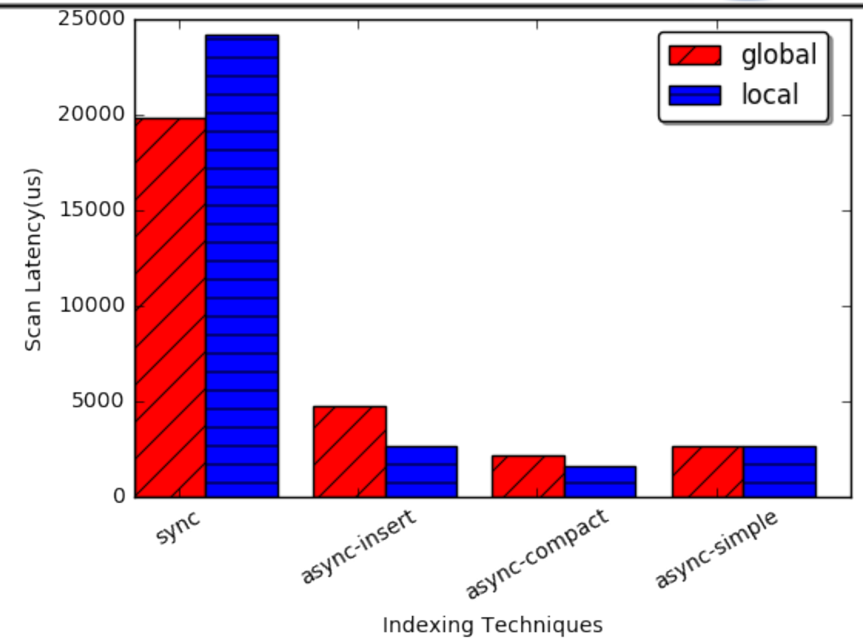




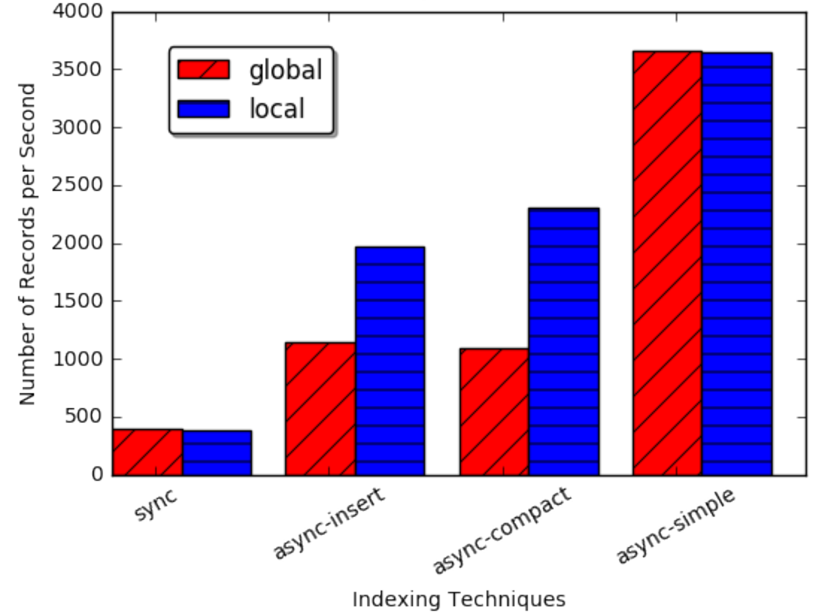
percentile / inconsistency window / indexing techniques	global				local			
	0.90	**0.95**	**0.97**	**0.99**	**0.90**	**0.95**	**0.97**	**0.99**
Async-insert	65922	70429	72243	74056	86015	90893	92846	94798
Async-simple	89010	93973	95959	97944	88854	93816	95801	97786

Evaluation- Scenario B





– Compared with update latency, scan latency is lower.
– The trend of performance of the indexing techniques is the same as scenario A
– The total performance of scenario B is better than scenario A

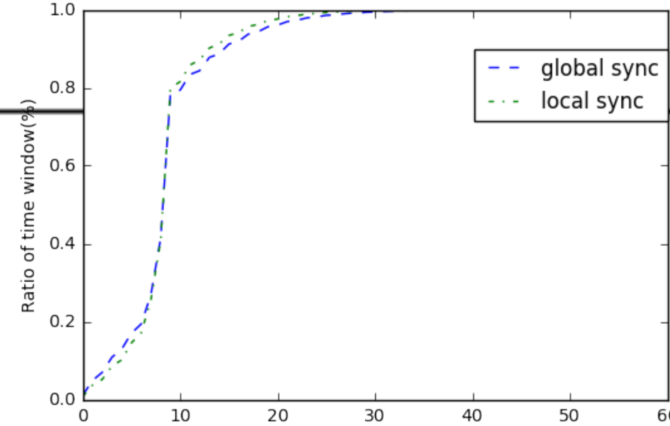


Evaluation- Scenario B

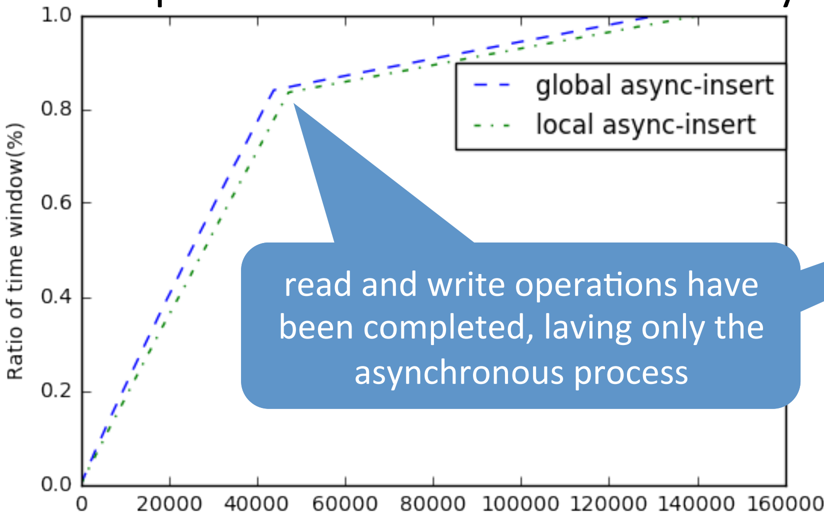
– Global and local have the similar consistency for the same indexing techniques.

– Sync has the highest degree of consistency.

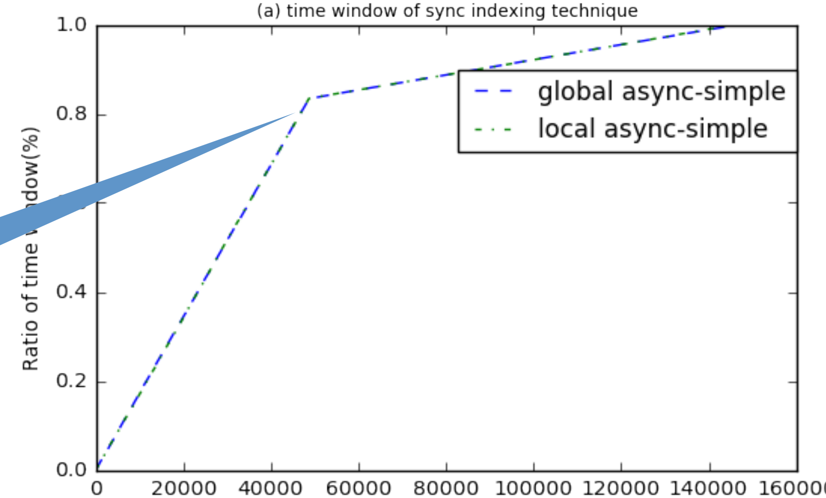
– Async-insert and async-simple indexing technique both show worse consistency.



(a) time window of sync indexing technique



(b) time window of async-insert indexing technique

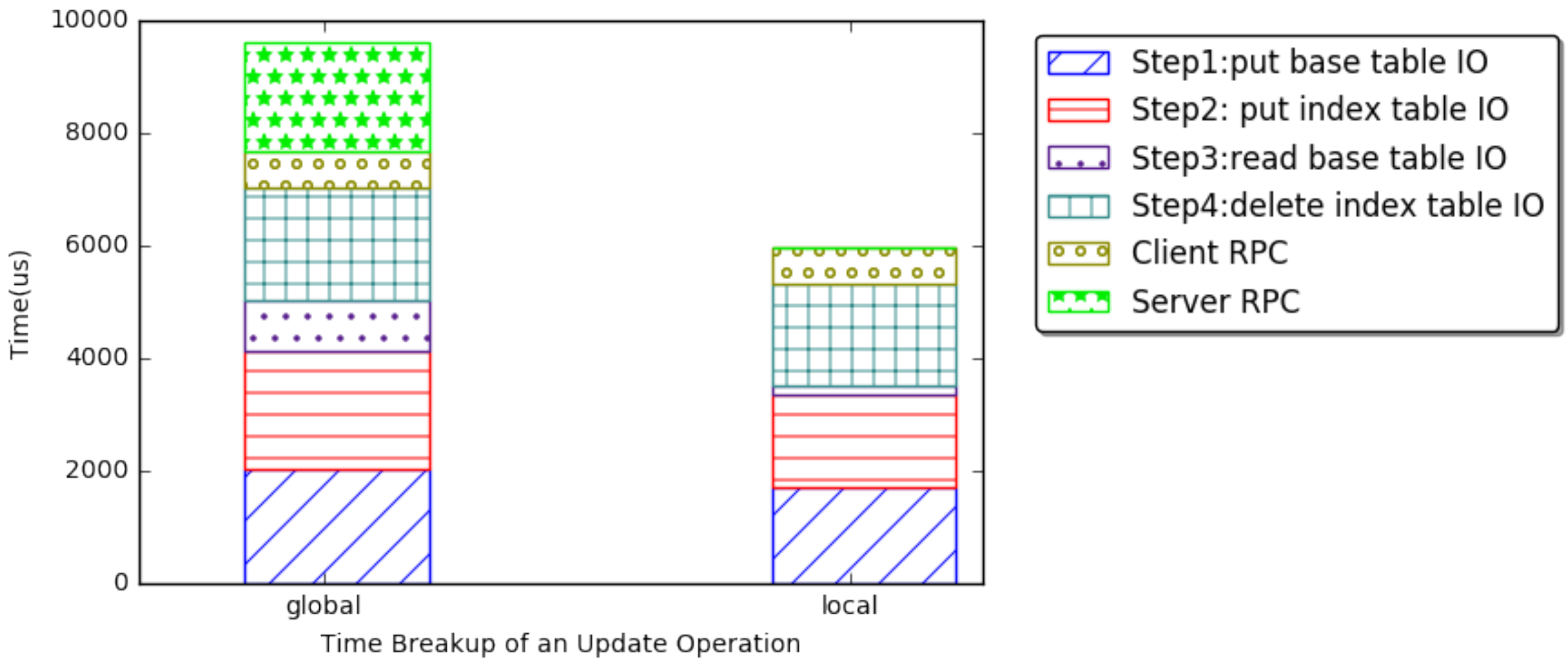


(c) time window of async-simple indexing technique

read and write operations have been completed, laving only the asynchronous process

percentile / inconsistency window / indexing techniques	global				local			
	0.90	**0.95**	**0.97**	**0.99**	**0.90**	**0.95**	**0.97**	**0.99**
Async-insert	76298	103843	114988	126019	83827	112353	123871	135404
Async-simple	87592	116876	128880	140760	87659	117052	128528	139925

Evaluation- time breakup



Local index have no sever RPC time compared with global index

Question 3

What can we learn form the above analysis ?

Conclusion

- Asynchronous indexing techniques are much better than synchronous indexing techniques in performance
- Async-simple is better than async-insert in performance while has similar inconsistency window with async-insert
- Comparing with the global indexing techniques, the local indexing techniques have no server RPC time

Feature / Item	Performance	Consistency
Async vs Sync	10x	worse
Async-simple vs Async-insert	1.86x-2.87x (local) 3.20x-6.26x (global)	similar
Local vs Global	better	similar

Conclusion

- Async-simple is better than async-insert in performance while has similar inconsistency window with async-insert ,why?

No.	Indexing Techniques	Description
2	Async-insert	Step1+Step2+async maintain
4	Async-simple	Step1+async maintain

- Although async-insert do step1 and step2 synchronization , but they are not atomic.

Future Work

- Why not async-simple local secondary indexing technique?
- Whether it can improve the consistency without decreasing the performance?

Related Work

- Index vs Replica
  - Consistency model [Lamport1979How], [Ahamad1995Causal], [Vogels2008Eventually], [Bailis2013Bolt], [Phansalkar2015Tunable]

Semantics	Secondary index database	Replicas system
Read	First reading the index table and then reading the base table.	Reading data from anyone replica.
Write	First writing data to base table and then writing data to index table or conversely.	No certain writing order between the replicas.

Related Work

- Indexing Optimization
  - *Concurrency Control* [graefe2012concurrency], [faleiro2015rethinking],
  - *Query Verify* [tan2014diff-index], [kejriwal2016slik]
  - *RDMA* [huang2012high-performance], [li2016accelerating]
  - *Double-Level Indexing* [wu2009an],[Cheng2014BF], [Sidirourgos2013Column]
  - *Domain-Specific Data* [Wang2014Lightweight]

Q&A

Thanks!