Spanner

A distributed database system

Presented by Yue Xia
Background

- Developed by Google initially as a key-value storage system
- Developers want traditional database features like query language
- Evolved to a full featured SQL system
- Now used by teams across many parts of Google and Alphabet
Overview

- Distributed computing
  - Architecture
    - Replication
    - Partitioning
    - Interleaving
  - Range Extraction
  - Distributed Union
  - Distributed Join

- Data storage
  - PAX
  - LSM Tree

- Conclusion and Discussion
Replication

- Multiple datacenters in different geographic locations
- Data replicated in each datacenter
- Run query on nearest datacenter
Partitioning

- Each datacenter has multiple servers
- Data row-range sharded (partitioned)
- Shards distributed across servers in each datacenter

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Alice</td>
<td>'A'</td>
</tr>
<tr>
<td>2</td>
<td>Eve</td>
<td>'A'</td>
</tr>
<tr>
<td>1</td>
<td>Carol</td>
<td>'B'</td>
</tr>
<tr>
<td>4</td>
<td>Bob</td>
<td>'C'</td>
</tr>
<tr>
<td>0</td>
<td>George</td>
<td>'D'</td>
</tr>
<tr>
<td>5</td>
<td>Fred</td>
<td>'D'</td>
</tr>
</tbody>
</table>
Interleaving

Parent & child tables interleaved and co-located

Customer Join Order only needs one scan of the interleaved table

<table>
<thead>
<tr>
<th>Customer Id</th>
<th>Customer Name</th>
<th>Order #</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer 1</td>
<td>Alice</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>order 1</td>
<td>$2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>order 2</td>
<td>$4</td>
</tr>
<tr>
<td>customer 2</td>
<td>Bob</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>order 1</td>
<td>$6</td>
</tr>
</tbody>
</table>
Query Execution

1. Go to the nearest datacenter

2. Extract key range

3. Run query only on shards covering the key range
Range Extraction – Goal

Given a query, we want to know:

- What shards to access

- What fragments of shards to access
  (seek into smaller key ranges instead of scanning the full shard)
Range Extraction - Filter Tree

Example:

scan Table
filter \( A=1 \) && ((\( B='a' \) && \( C=1 \)) || (\( B>a' \) && \( C=2 \))

Construct a tree according to the filter condition.
Range Extraction - Filter Tree

Example:

scan Table
filter $A=1$ && ((B='a' && C=1) || (B>'a' && C=2))

First find the range for A.

Assign leaf node an initial interval
Find the interval for each node from bottom to top.

AND is intersection and OR is union.

The range for A is [1,1]
Then find the range for $B$

- $A = 1$
  - $A: [1, 1]$
  - $B: [-\infty, \infty]$

- OR
  - $A: [-\infty, \infty]$
  - $B: ['a', 'a']$

  - AND
    - $B = 'a'$
      - $A: [-\infty, \infty]$
      - $B: ['a', 'a']$
    - C = 1
      - $A: [-\infty, \infty]$
      - $B: [-\infty, \infty]$

  - AND
    - $B > 'a'$
      - $A: [-\infty, \infty]$
      - $B: ('a', \infty]$
    - C = 2
      - $A: [-\infty, \infty]$
      - $B: [-\infty, \infty]$
Then C

Note that C’s range depends on B

A=1

AND

A:[1,1]
B:[-\infty, \infty]
C:[-\infty, \infty]

B='a'

A:[-\infty, \infty]
B:['a','a']
C:[-\infty, \infty]

C=1

AND

A:[-\infty, \infty]
B:['a','a']
C:[1,1]

C=2

AND

A:[-\infty, \infty]
B:['a','a']
C:[2,2]

B>'a'

A:[-\infty, \infty]
B:['a',\infty]
C:[1,1] if B='a', [2,2] if B>'a'

A:[-\infty, \infty]
B:['a',\infty]
C:[2,2]
Range Extraction - Query Rewrite

Rewrite filtered scan to self-join

scan Table
filter A=1 \&\& ((B='a' \&\& C=1) || (B>'a' \&\& C=2))
becomes:

Where the conditions in red are from the filter tree

Scan1(Table)  
Filter A in [1,1]
output @A

Scan2(Table)  
Filter A=@A
AND B in ['a',∞]
output @A,@B

Scan3(Table)  
Filter A=@A
And B=@B
And C = 1 if B=‘a’, C=2 if B>'a'
output columns of interest
Range Extraction – Execution

Scan1(Table)
Filter A in [1,1]
output @A

Output:
1

<table>
<thead>
<tr>
<th>record id</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>‘g’</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>‘a’</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>‘a’</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>‘ab’</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>‘b’</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>‘c’</td>
<td>2</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Range Extraction - Execution

Scan2(Table)
Filter A=@A AND B in ['a',∞]
output @A, @B

Output:
1, ‘a’
1, ’ab’,
1, ‘b’,

<table>
<thead>
<tr>
<th>record id</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>‘9’</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>‘a’</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>‘a’</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>‘ab’</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>‘b’</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>‘c’</td>
<td>2</td>
</tr>
</tbody>
</table>
Scan3(Table)
Filter A=@A
And B=@B
And C = 1 if B='a', C=2 if B>'a'
(output columns of interest
(output record_id if the table is sharded according to record_id)

Output:
3
Range Extraction - Query Rewrite

Join 3 tables.

Too slow?

Scan1(Table)
Filter A in [1,1]
output @A

Scan2(Table)
Filter A=@A
AND B in ['a',∞]
output @A, @B

Scan3(Table)
Filter A=@A
And B=@B
And C = 1 if B='a', C=2 if B>'a'
output columns of interest
Range Extraction - Actual Execution

<table>
<thead>
<tr>
<th>record id</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>‘g’</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>‘a’</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>‘a’</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>‘ab’</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>‘b’</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>‘c’</td>
<td>2</td>
</tr>
<tr>
<td>.......</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scan1(Table)
Filter A in [1,1]
output @A

Output 1 without accessing the data as A is fixed to 1.
Range Extraction - Execution

Scan2(Table)
Filter A=@A AND B in ['a',∞]
output @B

Seek the first record with A = 1 and B = ‘a’ instead of scanning the whole table.

Record 2 is found. Output 1, ‘a’

Then seek the next record with A = 1 and B != ‘a’, which is record 4. Output 1, ‘ab’ (skip records with B=’a’)
Range Extraction - Execution

Scan3(Table)
Filter A=@A
And B=@B
And C = 1 if B='a', C=2 if B>'a'
output columns of interest

Similar to Scan2, seek instead of scan. Can finger

<table>
<thead>
<tr>
<th>record id</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>'g'</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>'a'</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>'a'</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>'ab'</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>'b'</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>'c'</td>
<td>2</td>
</tr>
</tbody>
</table>

........
Range Extraction - Conclusion

- Filter Tree: find range for each column values.
- Rewrite filter to multiple self joins.
- Execute with seek instead of scan.
Distributed Union

- A new relational algebra operator
- Send subquery to each shard and concatenate results
Distributed Union

- Replace scan with distributed union of scan
  \[ \text{Scan(Table)} \rightarrow \text{DistributedUnion[shard} \subseteq T\](\text{Scan(shard)}) \]

- Pull distributed union above as many operations as possible. (push computation to each server)
Distributed Union

Some operations can be directly rewritten by:

\[ \text{Op}(\text{DistributedUnion}[\text{shard} \subseteq T](\text{Scan}(\text{shard}))) \rightarrow \text{DistributedUnion}[\text{shard} \subseteq T](\text{Op}(\text{Scan}(\text{shard}))) \]

Example:

- Basic operations like projection, filtering...
- Group by K or Ordered by K if sharded according to K
- Join of interleaved table
Distributed Union

Some need extra processing:

\[ \text{Op}(\text{DistributedUnion}[\text{shard} \subseteq T](\text{Scan}(\text{shard}))) \]

\[ \rightarrow \text{Op}\_\text{Final}(\text{DistributedUnion}[\text{shard} \subseteq T](\text{Op}\_\text{Local}(\text{Scan}(\text{shard})))) \]

Example:

- Top(5) can be done by finding the top 5 in each shard and then finding the top 5 among the results from all shards
Distributed Union - Optimization

- Multiple levels of distributed union
- On large shards, further parallelize between subshards
- Detect locally hosted shards and avoid remote call
Distributed Union – Optimization

- Range extraction:
  Extract key range
  Map key range to shards
  Send to min # of servers such that they contain all the required shards
  Only run query on required shards
Batched Distributed Join

- Join can also be distributed
- Send batches of left table to each shards
- Join batch with local shards
- Union
Batched Distributed Join - Optimization

- Select left table to fit in a batch
- Range extraction for each batch
- Construct the minimum batch to be sent to each shard
Data Storage - PAX

- Data stored in Partition Attributes Across (PAX) layout
- Records are horizontally partitioned in pages
- In each page all values of each attribute are grouped together
- Greatly improves cache performance
# Data Storage - PAX

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Alice</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>Bob</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>Carol</td>
<td>25</td>
</tr>
</tbody>
</table>

... ...

## PAX Page

0,1,2,...

Alice,Bob,Carol,...

15,20,25,...
- SELECT age WHERE age > 20
- Cache miss will cache the asked value and the values next to it

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Alice</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>Bob</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>Carol</td>
<td>25</td>
</tr>
</tbody>
</table>
Data Storage - LSM Tree

- Insert, update or delete would require rewriting the whole file.
Data Storage - LSM Tree

- One B-Tree on disk (fixed page size, 100% filled), and another in memory (smaller, no fixed block size).

- Updates are stored in the tree in memory.

- Merge two trees and write to disk when the tree in memory is large.
Data Storage - LSM Tree

- Write to disk

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
</tr>
</tbody>
</table>
```

Page 1

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
</tr>
</tbody>
</table>
```

Page 1

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
</tr>
</tbody>
</table>
```

Page 2

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
</tr>
</tbody>
</table>
```

Page 3
Data Storage - LSM Tree

- Write to memory

<table>
<thead>
<tr>
<th>Id</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

block 1

<table>
<thead>
<tr>
<th>Id</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
</tr>
</tbody>
</table>

block 2

<table>
<thead>
<tr>
<th>Id</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>

block 1

<table>
<thead>
<tr>
<th>Id</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

block 2

<table>
<thead>
<tr>
<th>Id</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
</tr>
</tbody>
</table>
Data Storage - LSM Tree

```
<table>
<thead>
<tr>
<th></th>
<th>21</th>
<th></th>
<th>34</th>
<th></th>
</tr>
</thead>
</table>
```

```
| 0.5 | 1.15 | 2.20 | 4.25 | ... |
```

```
<table>
<thead>
<tr>
<th></th>
<th>21</th>
<th></th>
<th>34</th>
<th></th>
</tr>
</thead>
</table>
```

```
| 3.10 |
```

Merge
Data Storage - LSM Tree

Write to new space
Data Storage - LSM Tree

```
* 21 * 34 *
```

```
0,5 3,10 1,15 2,20 4,25 ...
```

Repeat
Conclusion

- Replicated in datacenters
- Sharded and distributed among servers.
- Interleaved tables.
- Range Extraction
- Distributed Union and Distributed Join
- PAX and LSM Tree
Discussion - Range Extraction

Range extraction
- Rewrite filter to multiple self joins
- Each join outputs possible values of a column, last join outputs the range.
- Each join requires reading and scanning the table
- Some techniques like seeks to avoid scanning the entire table

Q:
Why don’t scan the table once and filter the records?
Discussion - Range Extraction

Q:
Why don’t scan the table once and filter the records?

A:
It hopes seeking in the table multiple times to be faster than scanning the full table once.

A full scan may be faster depending on the filter condition and table structure.

Should choose which method to use.
Q: Is it worth spending the extra time on range extraction?
Q: Is it worth spending the extra time on range extraction?

A: Saves time when table need to be scanned multiple times (like in join) and only a small portion of the table is useful.

However the cost of range extraction may outweigh the benefit in some cases.

Should decide whether to do range extraction.

Can give an approximate wider range instead of exact range.
Discussion - Range Extraction

Filter tree doesn’t work on some conditions. How to solve this?
For example: Filter A is odd, Regular Expression.

Solution:
Scan and filter instead.
Or ignore these conditions and give a wider range.
Thank You
References

[1] https://cloud.google.com/spanner/docs/whitepapers


[8] https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Row-major_order.html


