Apache S2Graph(incubating)
A Large Scale Distributed Graph Database
Table of Contents

1. What is S2Graph?
2. Why Did We Build It?
3. Architecture
4. Key Concepts
5. Performance
6. Use Cases
7. Future Work
What is S2GRAPH?
Property Graph Model: Vertices + Edges + Properties

S2Graph = Property Graph Model + Scalability + Fast CRUD Operations

Graph-processing layer atop HBase
Why Did We Build It?
About us

KakaoTalk

- #1 messenger in Korea
- Used by 93% of Korean mobile users
- Millions of requests per minute
- Messenger App → Do-Everything App

In South Korea, there's no escaping KakaoTalk. The country has a smartphone penetration rate of 73 percent, according to research firm EMarketer, and of those people, 93 percent use the mobile messaging app Kakao, says Nielsen.

Kakao expects its app will generate about $200 million in revenue this year, according to the company’s forecasts. That’s compared with $42 million last year, with a profit of $6.5 million.

But with little room left to grow in its home country, Kakao is looking elsewhere. That’s
Why We Got Into Graph Databases

30+ services at Kakao are powered by S2Graph

Graph Databases
- Relationship-Oriented
- Expressive yet Simple

Our Data
- Highly Connected
- Complex Relations
Technical Challenges

1. **Large graph constantly changing.**
   a. Total # of Edge: + 1 trillion and growing.
   b. Social Network: more than 10 billion edges, 200 million vertices, 50 million realtime update on social network.
   c. User activities(Click, Like, Share, Buy): 2.5 ~ 3 billion real-time incoming edges, 50 billion batch processed edges.

2. **Low latency for breadth first search traversal on connected data.**
   a. Peak graph-traversing query per minute: 4 million
   b. Average response time: 50 ms

3. **Update should be applied into query result in real-time.**

4. **Support for Dynamic Ranking logic**
   a. push strategy: hard to change data ranking logic dynamically.
   b. pull strategy: can try various data ranking logic
Existing solutions weren’t performant enough for our needs.

Especially,

1. Maintaining a mutable graph at scale was not supported. (i.e. Updates/Deletes were slow!)
2. Breadth First Search traversal was not fast enough.

So we built our own!
Architecture
Architecture Overview
Key Concepts
Data Model

Property Graph Model

- Not guarantee global uniqueness of your vertex Id.

Data Model for unique vertex Id guarantee.

- Service
  a. the top level abstraction, act like databases in traditional RDBMS.

- ServiceColumn
  a. names for vertices and a service can have multiple serviceColumns.
  b. ex) user_id, track_id, movie_id, ...

- Label
  a. represent relations between two serviceColumns. Think of it as a schema for edges.
  b. ex) friends, user_url_clicks, user_movie_play, ...

- Properties
  a. metadata linked to each vertex or each edge that can be queried upon later.
  b. ex) if your edge is user_music_play, then genre, artist, is_streaming, ... can be example properties on edge.

S2Graph encode model’s id with actual user provided vertex Id value to guarantee global uniqueness.
Introducing Snapshot Edges!

Imagine a Yelp-style application..

Reviews represented as edges

Update John’s review on McDonald’s
Iterate: $O(n) \Rightarrow$ Does not scale

Introducing Snapshot Edges! $\Rightarrow$ Enable access to indexed edges in constant time!!
Should yield same state regardless of deliver order of event.
It is guaranteed that request same set of requests multiple time, the final state would be same.
**Snapshot Edge: Consistency**

**Index Edge:**

- **Snapshot w/o Lock:**
  - John
  - McDonald’s
  - Properties of McDonald’s

- **Snapshot w/ Lock:**
  - John
  - McDonald’s
  - All properties of McDonald’s

---

<table>
<thead>
<tr>
<th>Actual Values in Storage</th>
<th>Thread 0: Update rating to 3.0</th>
<th>Thread 1: Update rating to 5.0</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index Edge</strong></td>
<td>State</td>
<td>Fetched Snapshot</td>
<td>State</td>
</tr>
<tr>
<td>Rating: 1.0</td>
<td>0</td>
<td>Original properties</td>
<td>0</td>
</tr>
<tr>
<td><strong>Snapshot Edge</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Snapshot Edge: Consistency**

**Index Edge:**

**Snapshot w/o Lock:**

**Snapshot w/ Lock:**

<table>
<thead>
<tr>
<th>Actual Values in Storage</th>
<th>Thread 0: Update rating to 3.0</th>
<th>Thread 1: Update rating to 5.0</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Edge</td>
<td>Snapshot Edge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rating: 1.0</td>
<td>Original properties</td>
<td>Original properties</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lock by thread 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

(* State Codes: 0 - trying, 1 - locked, 2 - updated, 3 - released)
# Snapshot Edge: Consistency

**Index Edge:**

**Snapshot w/o Lock:**

**Snapshot w/ Lock:**

<table>
<thead>
<tr>
<th>Actual Values in Storage</th>
<th>Thread 0: Update rating to 3.0</th>
<th>Thread 1: Update rating to 5.0</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Edge</td>
<td>Snapshot Edge</td>
<td>State</td>
<td>Fetched Snapshot</td>
</tr>
<tr>
<td>Rating: 1.0</td>
<td>Original properties</td>
<td>0</td>
<td>Original properties</td>
</tr>
<tr>
<td></td>
<td>Lock by thread 0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Rating: 3.0</td>
<td>Props updated by thread 0</td>
<td>2-3</td>
<td></td>
</tr>
</tbody>
</table>

(* State Codes: 0 - trying, 1 - locked, 2 - updated, 3 - released)
### Snapshot Edge: Consistency

**Index Edge:**

- John
- Rating: 1.0
- McDonald’s
- Properties of McDonald’s

**Snapshot w/o Lock:**

- John
- McDonald’s
- All properties of McDonald’s

**Snapshot w/ Lock:**

- John
- McDonald’s
- All properties of McDonald’s
- Lock by thread 0

### Actual Values in Storage

<table>
<thead>
<tr>
<th>Index Edge</th>
<th>Snapshot Edge</th>
<th>Thread 0: Update rating to 3.0</th>
<th>Thread 1: Update rating to 5.0</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rating: 1.0</td>
<td>Original properties</td>
<td>0</td>
<td>Original properties</td>
<td>0</td>
</tr>
<tr>
<td>Rating: 3.0</td>
<td>Props updated by thread 0</td>
<td>2-3</td>
<td>Props updated by thread 0</td>
<td>0</td>
</tr>
</tbody>
</table>

(* State Codes: 0 - trying, 1 - locked, 2 - updated, 3 - released)
**Snapshot Edge: Consistency**

**Index Edge:**
- John: Rating: 1.0
- McDonald’s: Rating: 3.0
- Properties of McDonald’s

**Snapshot w/o Lock:**
- John
- McDonald’s
- All properties of McDonald’s

**Snapshot w/ Lock:**
- John
- McDonald’s
- All properties of McDonald’s
- Lock by thread 0

<table>
<thead>
<tr>
<th>Acctual Values in Storage</th>
<th>Thread 0: Update rating to 3.0</th>
<th>Thread 1: Update rating to 5.0</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index Edge</strong></td>
<td><strong>Snapshot Edge</strong></td>
<td><strong>State</strong></td>
<td><strong>Fetched Snapshot</strong></td>
</tr>
<tr>
<td>Rating: 1.0</td>
<td>Original properties</td>
<td>0</td>
<td>Original properties</td>
</tr>
<tr>
<td></td>
<td>Lock by thread 0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Rating: 3.0</td>
<td>Props updated by thread 0</td>
<td>2–3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lock by thread 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rating: 5.0</td>
<td>Props updated by thread 1</td>
<td>2–3</td>
<td></td>
</tr>
</tbody>
</table>

(* State Codes: 0 - trying, 1 - locked, 2 - updated, 3 - released *)
Provide custom M/R job to serialize edge into HFile => No Work for RS!
**Supernode Handling**: Thundering Herd Problem

Cache **Future** instead of result!
1. Core use Asynchbase.
2. Rest server use play framework.
3. All of APIs are fully asynchronous not only for read but also write.
4. Provide blocking and non-blocking APIs for different kind of use cases.
5. Only admin operations such as create table use native blocking client.
Performance
Environment

1. HBase cluster
   20 Region Servers: 8-core, 32G memory, SSD

2. S2Graph App Server
   4-core, 16G memory

3. Test Data
   Total # of Edges: 100 billion (100 million rows x 1000 columns)
**Linear scalability**

- Benchmark Query: `src.out("friend").limit(100).out("friend").limit(10)`
- Total concurrency: 20 * # of app server
Varying width of traverse (tested with a single server)

- Benchmark Query: `src.out("friend").limit(x).out("friend").limit(10)`
- Total concurrency = 20 * # of app server
Different query path (different I/O pattern)

- All query touch 1000 edges.
- Each step’s limit is on x axis.
- Can expect performance with given query’s search space.
Write throughput per operation on single app server

InsertBulk operation

Latency

8000
16000
800000

Request per second
Write throughput per operation on single app server

Update (insert/update/delete) operation

Latency

Request per second
Statistics
  1. Total # of edges(activities + relations) >= 1 trillion
  2. Daily, new incoming edges in real-time >= 3 billion
  3. Daily, new edges that processed from batch process >= 50 billion
  4. Average query per minute >= 2 million. peak >= 4 million. under 50 ms.
  5. 40% queries are 3 step query, 40% are 2 step, 20% are 1 step.

Operations
  1. # of HBase region server = 40
  2. # of query server = 40
  3. # of write server = 20
Use Cases
Main Storage for user activities and relationships: Friends, clicks, purchases, likes, shares, comments, etc.
Real-time Recommendation (Spreading Activation):
1. Find items a user has reacted to (clicked, purchased..)
2. Find other users who reacted to the same items.
3. Find other items that those users reacted to.
Composite multiple queries via weighted sum

Ex) Ensemble of recommendation algorithms:
1. Item-based collaborative filtering (CF)
2. User-based CF
3. Matrix Factorization
4. Demographical MP items (for cold-start problem)
Native A/B (Bucket) Testing

1. Bucket = Query
2. Track performance of each bucket realtime: CTR, conversion rate..
Future Work

- More storage engines: Redis, RocksDB, Cassandra, and more
- More output formats: JSON, JAVA, Thrift, Protobuf, and more
- Graph computing frameworks: Apache TinkerPop
- Usability: Web UI, better API
- Build a FUN COMMUNITY!!
Community

- Checkout: https://s2graph.incubator.apache.org/
- Subscribe: dev-subscribe@s2graph.incubator.apache.org, users-subscribe@s2graph.incubator.apache.org