Graph Processing with Apache Tinkerpop on Apache S2Graph(incubating)

TABLE OF CONTENTS

- BACKGROUND
- TINKERPOP3 ON S2GRAPH
- UNIQUE FEATURES OF S2GRAPH
- BENCHMARK
- FUTURE WORK

BACKGROUND

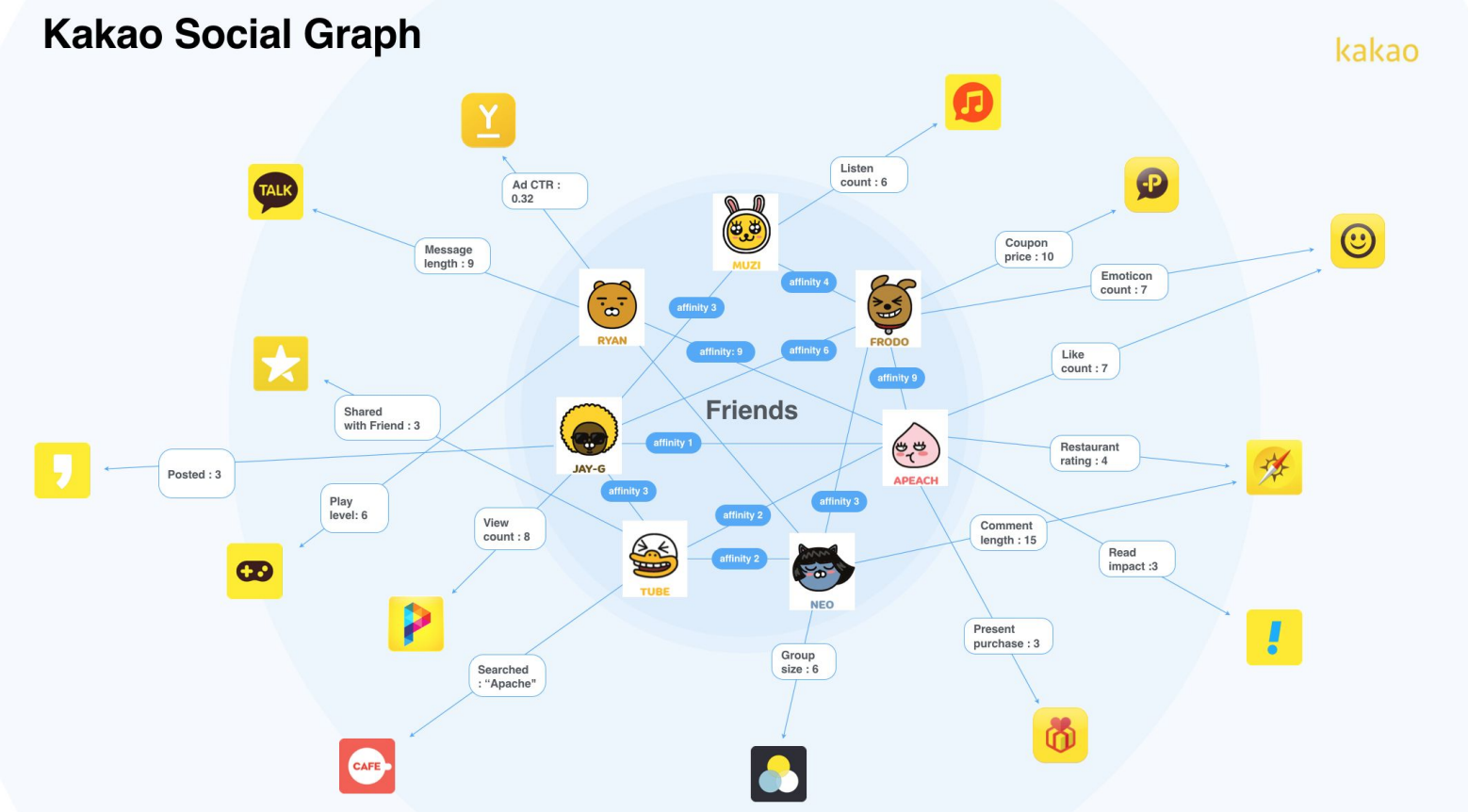
BACKGROUND - OUR GRAPH

The most interesting graph we have is mixed pattern across multiple domains.

- **MOBILE MESSENGER** - friends relations.
- **SEARCH ENGINE** - search history, click history
- **SOCIAL NETWORK** - friends relations
- **CONTENTS DISCOVERY** - Lots of interaction among User, Music, Micro Blog, News, Movie ...
- **LOCATIONS** - check-in, check-out

Many orthogonal domains(+30) are connected based on User object.
Graphs that connect one type of node with other kinds of nodes.

BACKGROUND - IN ONE PICTURE



BACKGROUND - OUR OPERATIONS

- **PERSONALIZED SUBGRAPH** - give me subgraph starting from one vertex.
  - What is the post, pictures, musics that my friends are clicking now.
  - What others who interacted on same
  - Need to be fast, and efficient.

- **TARGETING AUDIENCE** - give me all users who meet this condition.
  - How many people who searched apache within a week and also visited last apachecon achieve, also live in south korea.

PERSONALIZED SUBGRAPH = OLTP
TARGETING AUDIENCE = OLAP

BACKGROUND - MOTIVATION

THERE IS NO SINGLE SILVER BULLET YET

- Don’t think it is possible to run OLTP and OLAP on same storage in production.
  - BlockCache eviction, Excessive disk I/O with heavy OLAP query.

- Not only interested in Graph algorithm, but also interested in basic analytics on user interactions.
  - Not just pagerank, shortest path, connected component.
  - Find out what is average number of friends who searched apach and visited last apache con achieve

S2Graph wants to be a strong OLTP graph database, not Graph Processor.
However provide tools to integrate S2Graph into other OLAP systems.

BACKGROUND - MOTIVATION

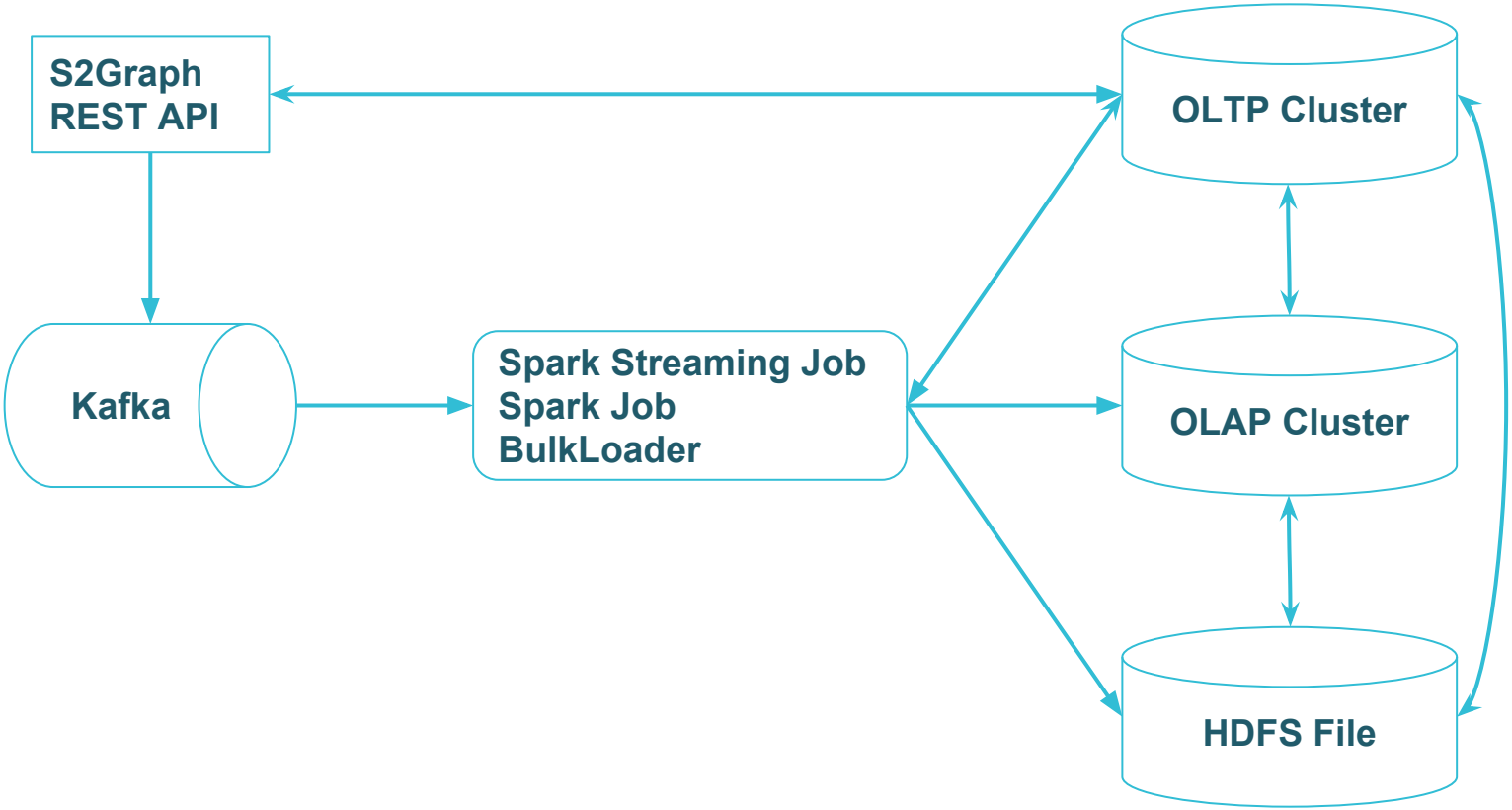
S2Graph community has been working on providing the following tools.

- Store every requests into Apache Kafka.
- ~~Provide Replication on Analytic HBase Cluster(possible, but will be deprecated soon).~~

S2Graph has a loader subproject that manage following tools.

- Append stream in Kafka to HDFS directly as optionally Graph JSON format.
- Bulk Loader to upload large graph without performance penalty in production.
- ETL environment that can join metadata from S2Graph on incoming stream in Kafka.
- Transfer stream in Kafka to star schema by joining metadata at S2Graph.

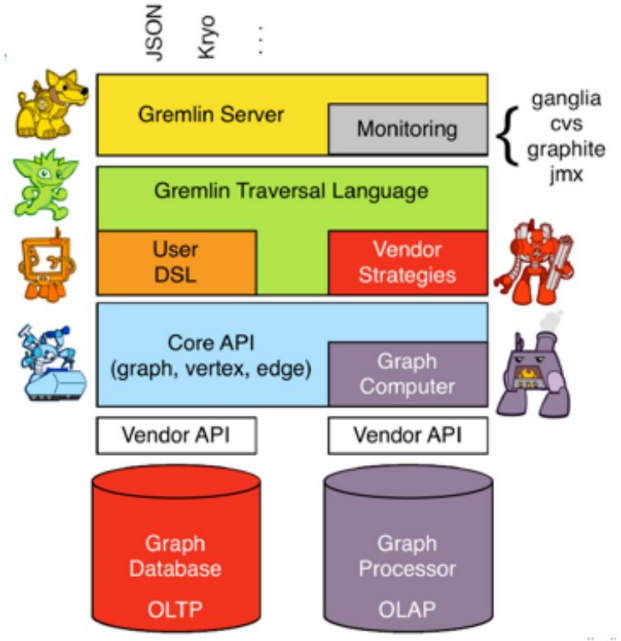
ARCHITECTURE - BOTH FOR OLTP AND OLAP



TINKERPOP3 ON S2GRAPH

TINKERPOP3

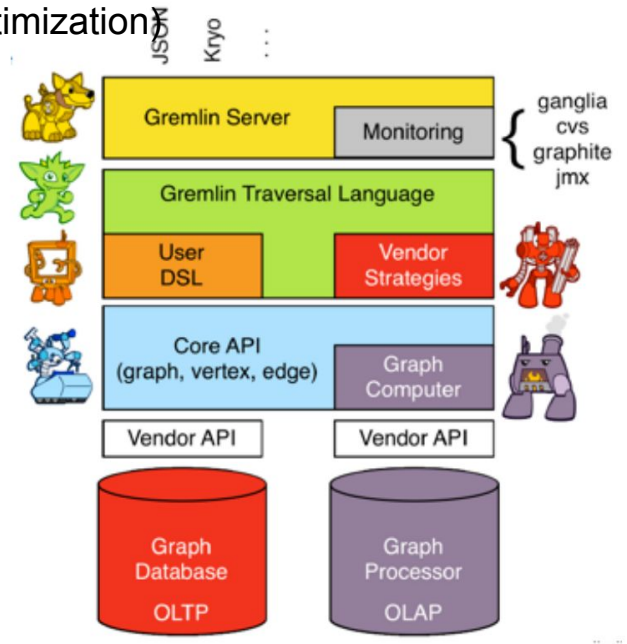
- A standard, vendor-agnostic graph API
- A standard, vendor-agnostic graph query language Gremlin
- OLTP and OLAP engines for evaluating query
- Sample Data Server
- Reference TinkerPop 3 graph implementation
- Reference command line shell Gremlin shell
- Reference visualization Gephi integration.



TINKERPOP3 ON S2GRAPH

Implementing Gremlin-Core

- OLTP
  - **Structure API** - Graph/Vertex/Edge, etc.
  - **Process API** - TraversalStrategy(vendor specific optimization)
- IO
  - **GraphSON I/O Format**



TINKERPOP3 ON S2GRAPH

Implementing Gremlin-Core: S2GRAPH-72

- **OLTP**
  - **Structure API** - Graph/Vertex/Edge, etc.
    - Done once under S2GRAPH-72 in scala.
    - Working on Java Client for better interpolation.
    - Planned to be included in 0.2.0 release.
  - **Process API** - TraversalStrategy
    - Block on every step even though S2Graph's Step's are all async.
    - Need to discuss tp3 to provide async Step.
    - Research on how to transfer gremlin query to S2Graph's optimized query.
    - Maybe possible on 0.3.0 release.
- **IO**
  - **GraphSON I/O Format**
    - Personally, tried out but never discussed, reviewed formally.
    - Planned to be included in 0.2.0 release.

TINKERPOP3 ON S2GRAPH

BASIC OPERATION - CREATE/GET

```
// addVertex
s2.traversal().clone().addV("serviceName", "myService", "columnName", "myColumn", "id", 1).next()

// getVertices
s2.traversal().clone().V("myService", "myColumn", 1)

// addEdge
s2.traversal().clone().
      addV("id", 1, "serviceName", "myService", "columnName", "myColumn").as("from").
      addV("id", 10, "serviceName", "myService", "columnName", "myColumn").as("to").addE("myLabel")
      .from("from")
      .to("to")
      .next()

// getEdge
s2.traversal().clone().V("myService", "myColumn", 1).outE("myLabel").next(10)
```

TINKERPOP3 ON S2GRAPH

SETUP

```
val config = ConfigFactory.load()
val g: S2Graph = new S2Graph(config)(ExecutionContext.Implicits.global)

val testLabelName = "talk"
val testServiceName = "kakao"
val testColumnName = "user_id"

val vertices: java.util.ArrayList[S2VertexID] =
  Arrays.asList(
    new S2VertexID(testServiceName, testColumnName, Long.box(1)),
    new S2VertexID(testServiceName, testColumnName, Long.box(2)),
    new S2VertexID(testServiceName, testColumnName, Long.box(3)),
    new S2VertexID(testServiceName, testColumnName, Long.box(4))
  )
```

TINKERPOP3 ON S2GRAPH

SETUP

```
val vertices = new util.ArrayList[S2VertexID]()

ids.foreach { id =>
  vertices.add(new S2VertexID(testServiceName, testColumnName, Long.box(id)))
}

g.traversal().V(vertices)
 .outE(testLabelName)
 .has("is_hidden", P.eq("false"))
 .outV().hasId(toVId(-1), toVId(0))
 .toList
```

TINKERPOP3 ON S2GRAPH

BASIC 2 STEP QUERY - VertexStep is now blocking.

```
// S2Graph Tinkerpop3 query
g.traversal().V(vertices)
 .out(testLabelName)
 .limit(2)
 .as("parents")
 .in(testLabelName)
 .limit(1000)
 .as("child")
 .select[Vertex]("parents", "child")
```

```
# S2Graph Query DSL
{
        "srcVertices": [{
        "serviceName": "kakao",
        "columnName": "user_id",
        "ids": [1, 2, 3, 4]
    }],
    "steps": [
        [{
            "label": "talk",
            "direction": "out",
            "offset": 0,
            "limit": 2
        }],
        [{
            "label": "talk",
            "direction": "in",
            "offset": 0,
            "limit": 1000
        }]
    ]
}
```

TINKERPOP3 ON S2GRAPH

FILTEROUT QUERY

```
val excludeIds =
  g.traversal()
    .V(new S2VertexID("kakao", "user_id", Long.box(1)))
    .out("talk")
    .limit(10)
    .toList
    .map(_.id)

val include =
  g.traversal()
   .V(new S2VertexID("kakao", "user_id", Long.box(2)))
   .out("talk")
   .limit(5)
   .toList

include.filter { v => !excludeIds.contains(v.id()) }
```

```
# S2Graph Query DSL
{
    "limit": 10,
    "filterOut": {
        "srcVertices": [{
            "serviceName": "kakao",
            "columnName": "user_id",
            "id": 2
        }],
        "steps": [{
            "step": [{
                "label": "talk",
                "direction": "out",
                "offset": 0,
                "limit": 10
            }]
        }]
    },
    "srcVertices": [{
        "serviceName": "kakao",
        "columnName": "user_id",
        "id": 1
    }],
    "steps": [{
        "step": [{
            "label": "talk",
            "direction": "out",
            "offset": 0,
            "limit": 5
        }]
    }]
}
```

WHEN WILL THIS AVAILABLE?

Implementing Gremlin-Core: S2GRAPH-72

- v0.2.0: will include
  - Structure API
  - GraphSON IO Format
- v0.3.0: may include
  - Process API: optimization from implementation
  - All tools for integration with OLAP system

UNIQUE FEATURES OF S2GRAPH

PARTITION

Storage Backend(HBase) is responsible for data partition, but provide followings.

- Pre-split level per Label.
  - Pre-split act as top level partition range on hierarchy.
  - \x19\x99\x99\x99 3332 : partition 0, responsible for murmur hash range from 0 ~ Int.Max / 2
    3332 L\xCC\xCC\xCB : partition 1, responsible for murmur hash range from Int.Max / 2 ~ Int.Max
  - \x19\x99\x99\x99 1132\xdf : partition 0
    1132\xdf 3332 : partition 0-1
    3332 L\xCC\xCC\xCB : partition 1

HBase, Cassandra can provide partitioning, but Redis, RocksDB, Postgresl, Mysql does not support this, so currently limited to single machine with these storage backend.

Need to discuss if it is S2Graph’s role to maintain partition metadata.

ID MANAGEMENT

Instead of convert user provided Id into internal unique numeric Id, S2Graph simply composite service and column metadata with user provided Id to guarantee global unique Id.

- **Service - the top level abstraction**
  - A convenient logical grouping of related entities
  - Similar to the database abstraction that most relational databases support.
- **Column - belongs to a service.**
  - A set of homogeneous vertices such as users, news articles or tags.
  - Every vertex has a user-provided unique ID that allows the efficient lookup.
  - A service typically contains multiple columns.
- **Label - schema for edge**
  - A set of homogeneous edges such as friendships, views, or clicks.
  - Relation between two columns as well as a recursive association within one column.
  - The two columns connected with a label may not necessarily be in the same service, allowing us to store and query data that spans over multiple services.

DESIGN TO BE USED WITH EXTERNAL SYSTEM

User provided Id is very important because it all relate to followings.

- Easy to work with existing legacy solution.
  - It's possible to migrate only relation table(edge), not object table(vertex).
  - Most cases, Object usually stored in RDBMS, because it is already there.
  - Graph Database is more about replacement of relation table, so ability to store Primary Key of object in existing table becomes extremely important to work with legacy.
  - S2Graph focus on edge, rather than vertex because of this reason.

- Mutations on graph is a stream, not one time only batch.
  - Need two Id lookup to insert edge.
    - Cache can help us, or may not depends on your data.
    - Lots of resource on client side will be required since client need to wait for vertex Ids for each edge insertion.
  - No read operation for edge insertion required, since all necessary data to insert edge is provided from user. Much performant and efficient on streaming environment.

IDEMPOTENCY

Guarantee the same eventual state for a given set of request-timestamp combinations, regardless of the actual arrival order or of the requests.

- To ensure a consistent eventual state, we use the following read-modify-write steps with the user-provided timestamp information for versioning.
    - Step 1 Read the latest state of the snapshot edge.
    - Step 2 Update the snapshot edge with the most up-to-date data and build the mutations needed to be made.
    - Step 3 Apply the mutations in the storage.

Since every property in the snapshot edge includes its timestamp, S2Graph can compare this with the request's timestamp to keep only the most recent information and drop any old properties.

Desired feature to work with stream.

OPTIMISTIC CONCURRENCY CONTROL

To guarantee idempotency, concurrent mutations on the same edge must not happen.

- The typical resolution to this synchronization problem is to provide users with the edge-level transaction.
  - Users write the retry logic manually.
  - Makes the lock contention to happen more frequently.

S2Graph uses an optimistic concurrency control where the system never acquires a lock at the read time, but rather resolves any conflict at the write time

- Use CompareAndSet API of HBase, or similar atomic operations of other backends.

Help users write a fully asynchronous application

- S2Graph's implementation of the optimistic locking and automatic retries is fully event-based, never makes a user thread block.
- Users have to start a transaction and commit/rollback typically in a try/catch block with transaction.

FULLY ASYNCHRONOUS CACHE HANDLING

Fully Asynchronous Cache Handling: A supernode is a vertex attached to a disproportionately high number of edges.

- Popular items or users

Supernode result in hot-spots

- massive requests hitting a small number of region servers
- preventing the system from ensuring the scalability.

S2Graph implements a lock-free result cache for efficiently handling supernodes

- backend access happens only once per JVM even with multiple requests for a single supernode.

Many requests goes straight to the backend without hitting the cache until the first one is completed and get loaded into cache.

- caching the promises of the asynchronous data access operations, in addition to the actual data.

BULK INSERT

Challenge: Large dataset while retaining the normal workload unaffected.

Native bulk loading feature provided by HBase.

1. Build HFile using S2Graph's serialization API with Apache Spark in a seperate analytics cluster.
2. Transfer them to the production cluster.
3. Instantly imported as HBase tables

The total cost on the production cluster caused is simply the cost for HFile transfer.

CUSTOMIZE STORAGE SCHEMA - WIDE or THIN ROW

Cassandra user prefer wide row.

HBase user prefer thin row.

It is up to user which schema to use and user can provide their own schema.

Default is Thin row schema since default storage backend is HBase.

BUILT-IN A/B(BUCKET) TESTING

Support a quantitative comparison between two or more strategies for business decision.

It is possible to set up multiple buckets with different querying logics and parameters.

- Whenever data exposed into user and user click, insert feedback as edge.
- Track the real-time performance of each bucket.

BENCHMARK

BENCHMARK

- Use SocialSensor's benchmark which uses a common interface for graph operations whose implementations for the three baseline databases are already provide.

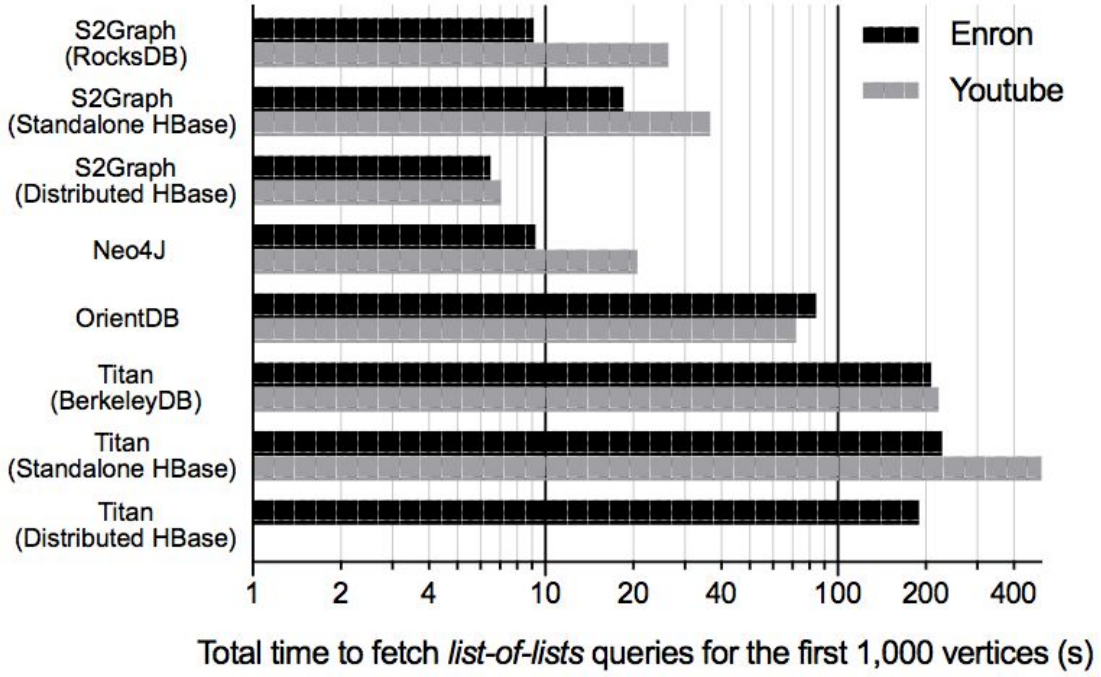
- Benchmark workload: Friend of Friend query on available datasets without limit paramter on the first 1000 start vertices.

- Dataset
  - Enron email dataset: 36,693 vertices and 183831 edges
  - Youtube social network dataset: 1134890 vertices and 2987624 edges.

- System versions
  - Neo4j 3.0.3
  - OrientDB 2.2.5
  - Titan 1.0.0

- Test environment
  - Single mode JVM 8 with 16GB heap space, Distributed mode 5 region servers running 16GB machine with 8G heap space.

BENCHMARK


S2Graph
(RocksDB)
S2Graph
(Standalone HBase)
S2Graph
(Distributed HBase)
Neo4J
OrientDB
Titan
(BerkeleyDB)
Titan
(Standalone HBase)
Titan
(Distributed HBase)
Enron
Youtube
1
2
4
10
20
40
100
200
400
Total time to fetch *list-of-lists* queries for the first 1,000 vertices (s)

FUTURE WORK

FUTURE WORK

- v0.2.0-incubating release planned on the end of Dec.
  - This include Tinkerpop OLTP structure package implementation in Java.
  - Also include IO format to transfer S2Graph's serialization into GraphSON format.
  - More Storage Backend support.

- v0.3.0-incubating release may comes one or two month after v0.2.0.
  - May include Traversal Strategy.

- Overhaul website contents.