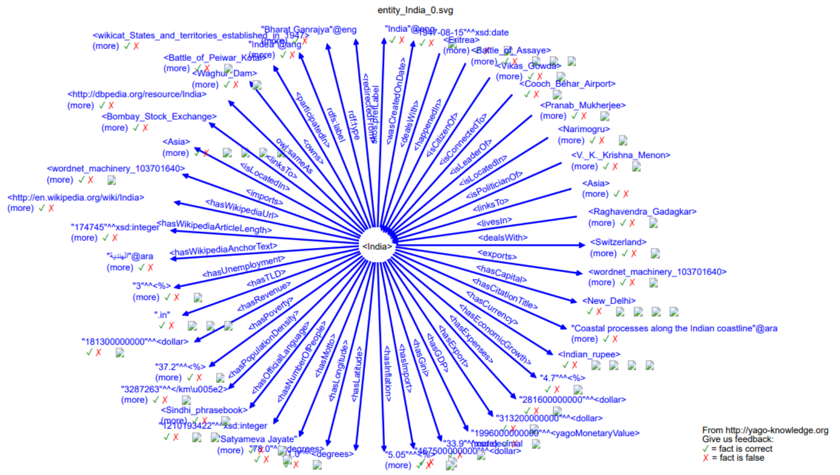
Distributed Knowledge Graph Querying on Edge and Cloud

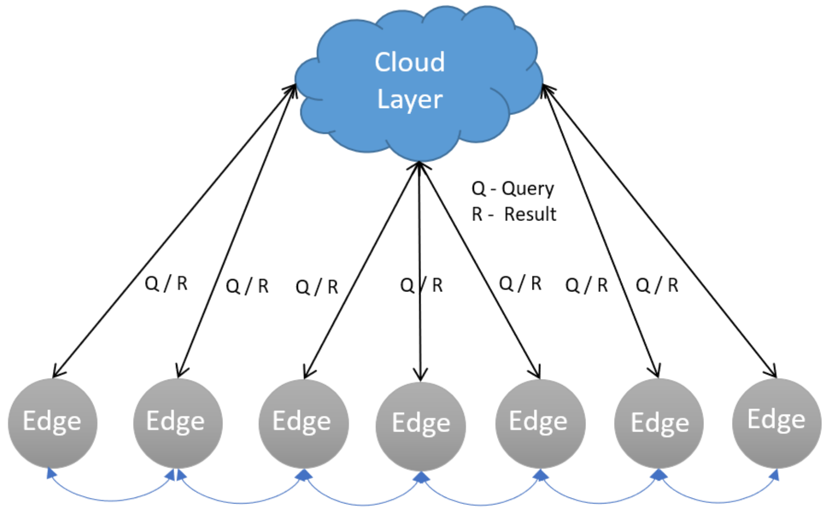
DS 256 – Project Final Presentation

Shriram R

Motivation

- Knowledge Graphs are large
- Graph querying problems are hard
- IoT – Heterogenous Structure
  - Different Compute, Storage & Network Capacity
- Needs lightweight and low latency solution





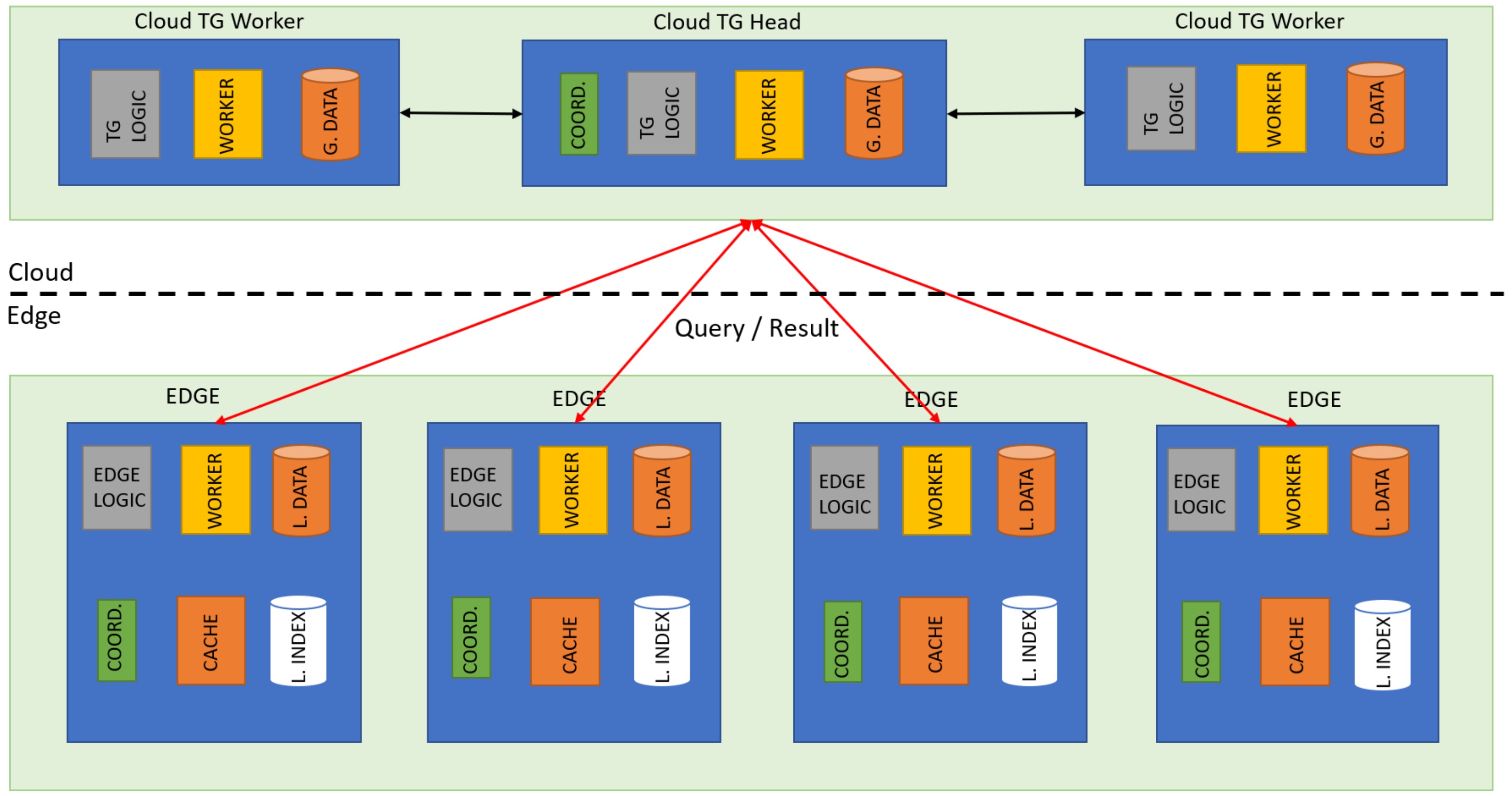
Related Work

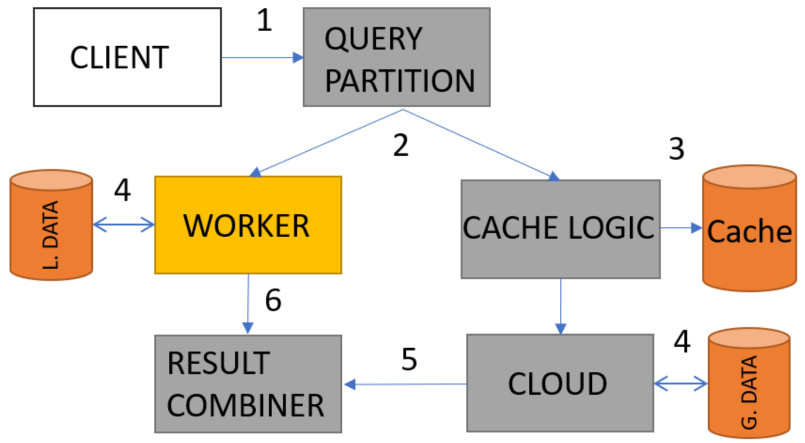
- Graph Processing
  - Pregel [2]
  - Giraph [3]
  - GraphX [4]
  - Trinity [5]
  - Quegel [13]
- Graph Caching
  - GC [6]
- Graph Indexing
  - C-Tree [7]
  - Views [8]
  - FERRARI [9]
  - GraphS [10]
- Graph Databases
  - GoDB [11]
  - TinkerGraph
  - Neo4j
  - Dgraph
  - Titan

Key Contributions

- Graph Caching
  - Cache Knowledge graph in Edge layer
- Query Partitioning
  - Partition input graph query into local and remote
- Result Generation
  - Combine results from local and remote server into a correct result
- Different Query Types
  - Vertex, Edge, Path search and reachability
- Result Caching
  - Cache results from remote layer with cache management

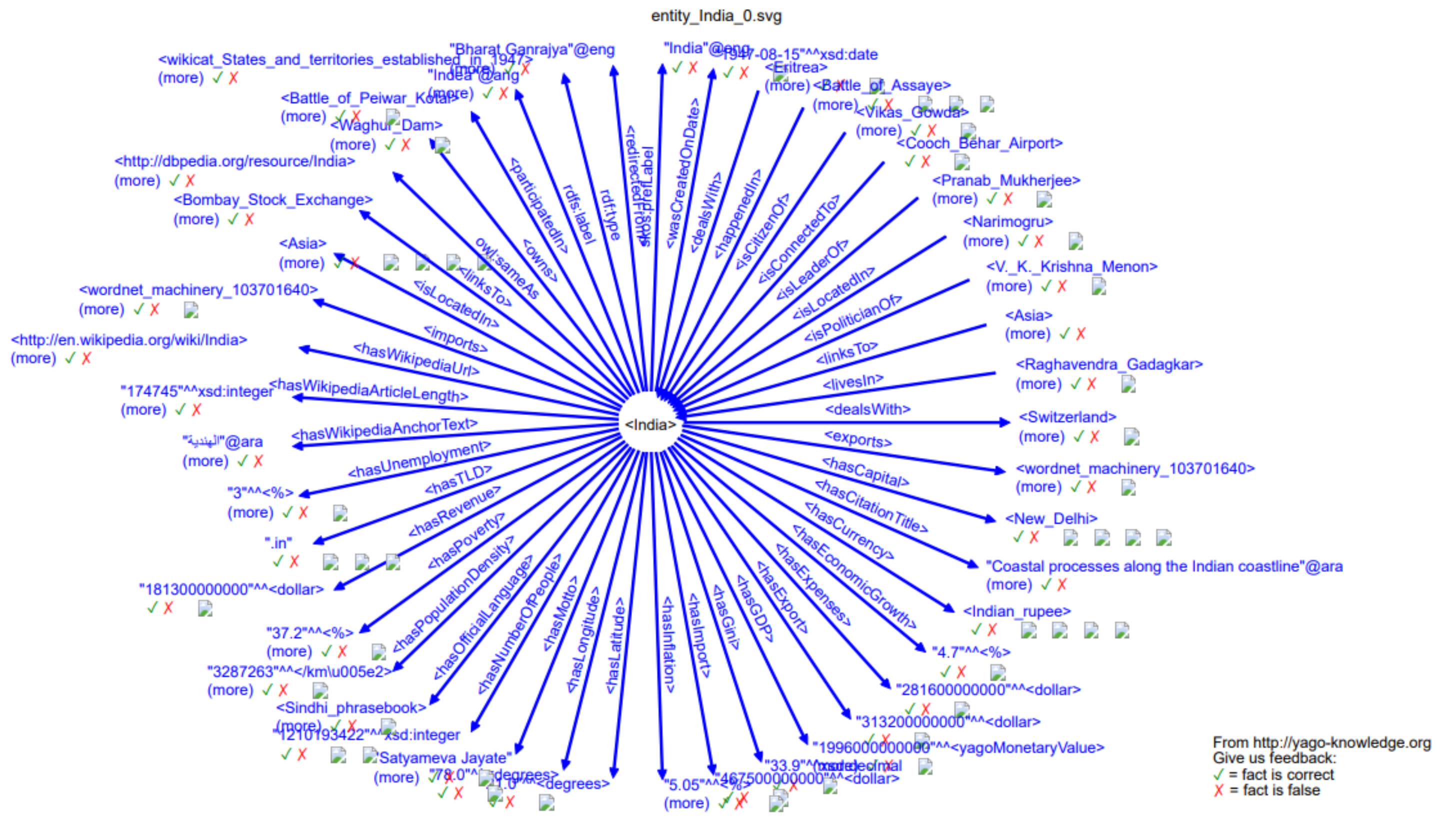
System Architecture





Graph Caching

- Associate an entity with the edge device E.g. <India>
- Store the k-hop subgraph around the entity as local graph
- Parameter k depends on storage and compute capacity



Query Processing

- Reachability
  - Type 1 - Entirely contained within edge (local graph)
  - Type 2 - Crosses between edge and remote (local + remote graph)
  - Type 3 - Entirely contained within remote (remote graph)
- Steps
  1. Local Query is fired to check for Type 1 existence
  2. If no result in previous step, remote queries are fired with source as Cut Vertices for Type 2
  3. Local queries are fired with source unchanged and target as Cut Vertices
  4. Remote query is also fired with original source for Type 3

Query Processing

- Path Query
  - *N* vertex and *N-1* edge predicates
  - *Null* in query is wildcard
  - If path length < k (hops)
    - Fire local and remote query
  - Else
    - Fire only remote query
  - Union paths from local and remote query for the result

```
// Path Query Example
"type": "path_search",
"filter": {
"vertices": ["Mahatma_Gandhi", null, "Subhash_Agarwal"],
"edges": ["isCitizenOf", null]
 }
}
```

Result Caching

- Edge device caches query results from remote server
- Cache size in terms of bytes is parameterized
- Whenever a remote query is fired, cache is checked for hit
- New query is inserted with key as query and value as result
- FIFO Replacement policy
- Local queries are not cached

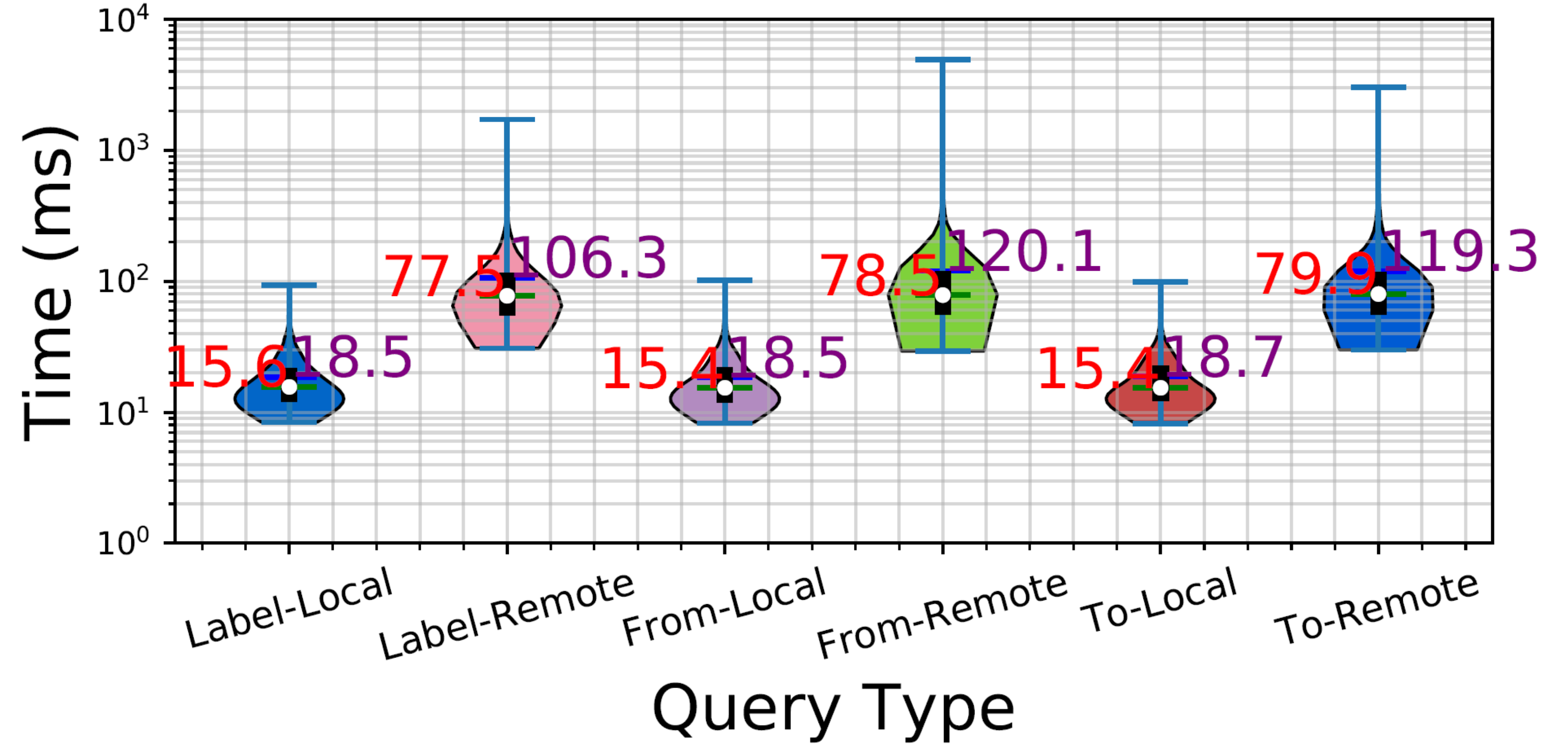
Implementation

- TinkerGraph used for both Cloud and Edge
- Edge logic written in *Python*
- *GremlinPython* package used for communication
- *GraphML* format used to persist data

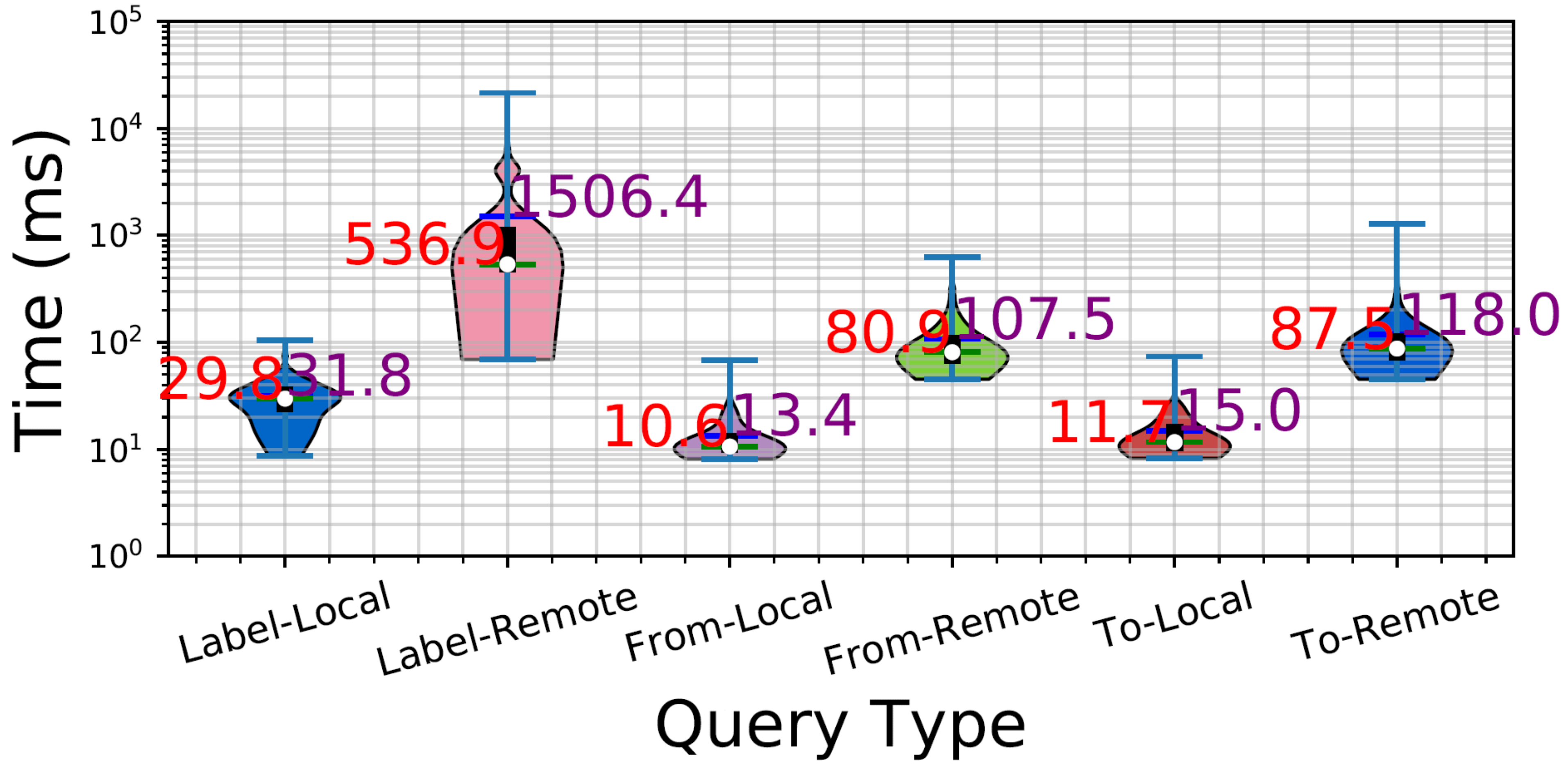
Experiments

- Remote Graph database
  - 1 node in-memory spawned TinkerGraph in Rigel Head Node
  - 32 core AMD Opteron with 128 GB RAM
  - Centos 7
- Edge Layer
  - 4 core, 1GB RAM container
  - Ubuntu 18.04 LTS
  - Latency: 5ms & Bandwidth: 100 MBps
- Dataset
  - YAGO minimal knowledge graph
  - 14977 vertices, 18845 edges and 1400 total attributes
  - Local graph centred on <India> with 2 hops

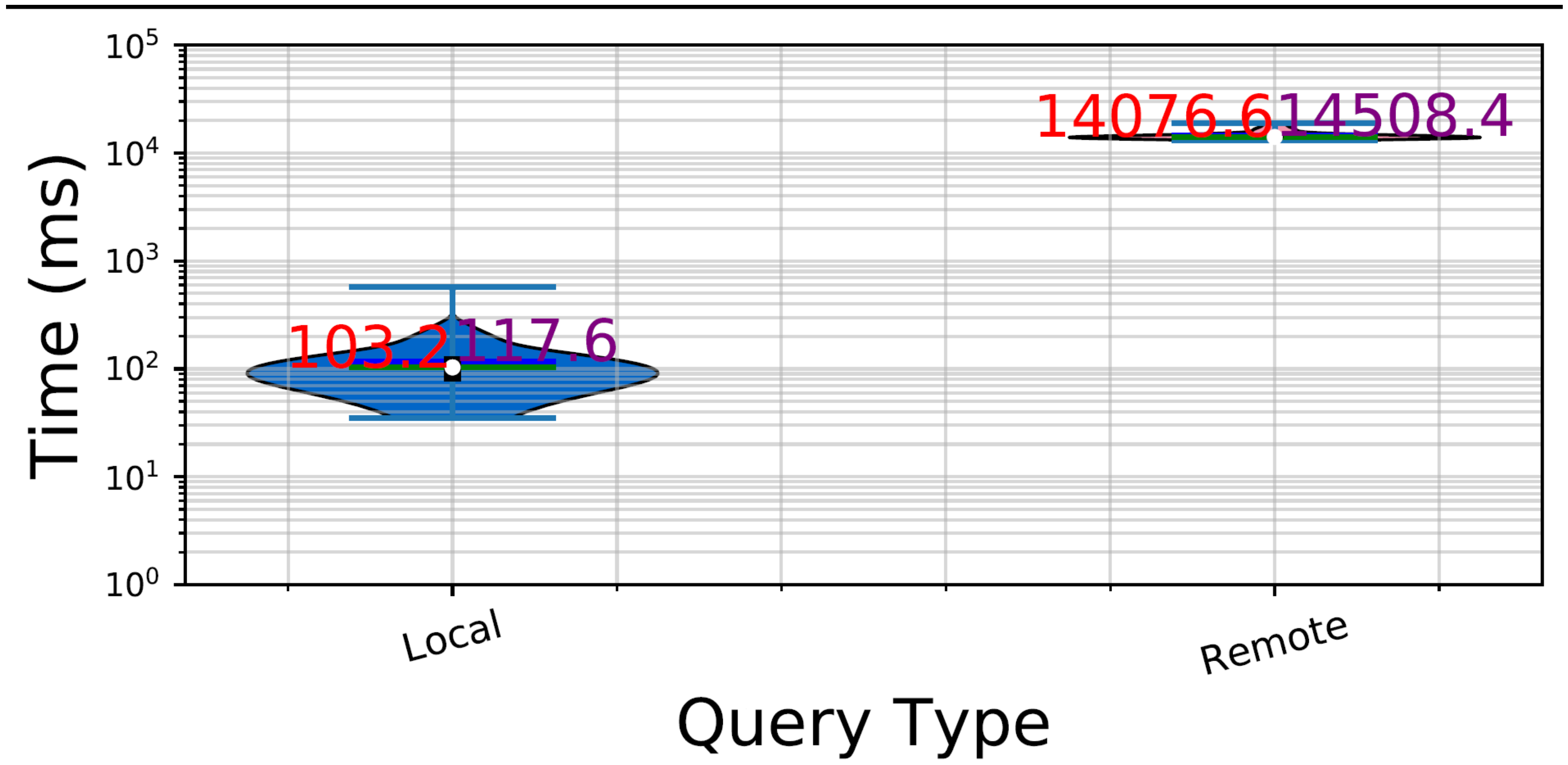
Experiments – Vertex Search



Experiments – Edge Search



Experiments – Reachability



Challenges & Future Work

- Challenges
  - Integrating *GoDB* with the project
  - Setting up distributed environment for *TinkerGraph*
  - Difficulty in setting up large cluster for experiments
  - ETL for the dataset
- Future Work
  - Evaluate caching and all queries in large setup
  - Index the graph on the edge layer
  - Integrate with different datasets
  - Implement machine learning algorithms using this system

Original Proposal & Accomplishments

- Midterm
  - Graph Caching
  - Query Partitioning
  - Combining results
  - Vertex, Edge search and Reachability
  - Experiments
- Final
  - Caching of remote results
  - Support for path queries
  - Indexing
  - Experiments
- Stretch Goals
  - Subgraph isomorphism queries
  - Support for Graph updates