Entity Centric Indexing in Rya

Rya Working Group

April 20, 2016

Problem Statement

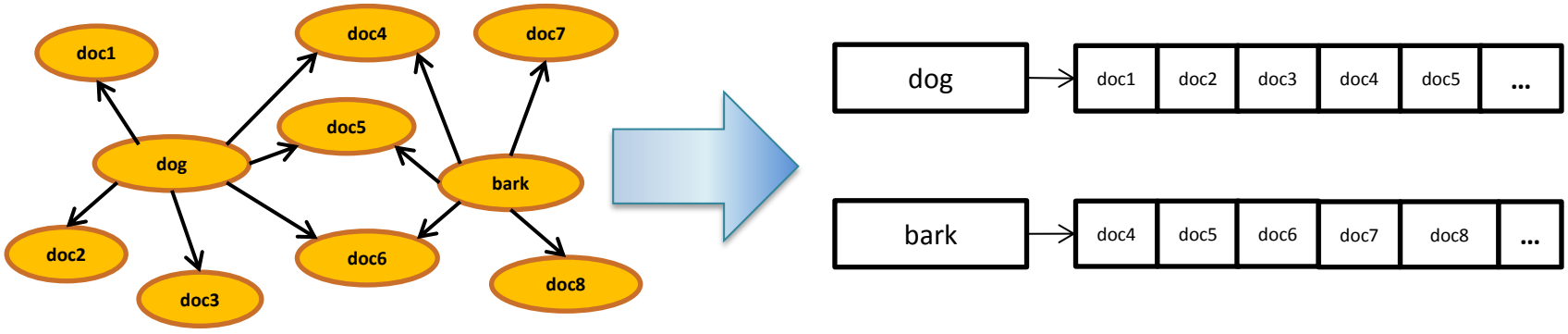
- Find all documents in a datastore that contain a specified collections of terms.
- The following SPARQL query asks for all documents that contain the terms “dog” and “barks”.

```
SELECT ?X
WHERE {
    ?X contains “dog” .
    ?X contains “barks” .
}
```

Adjacency Lists

One possible approach:

- View docs and terms as a graph, with edges drawn from a term to any document which contains that term
- Efficiently represent graph as a collection of adjacency lists
- Finding common documents reduced to finding intersection of lists

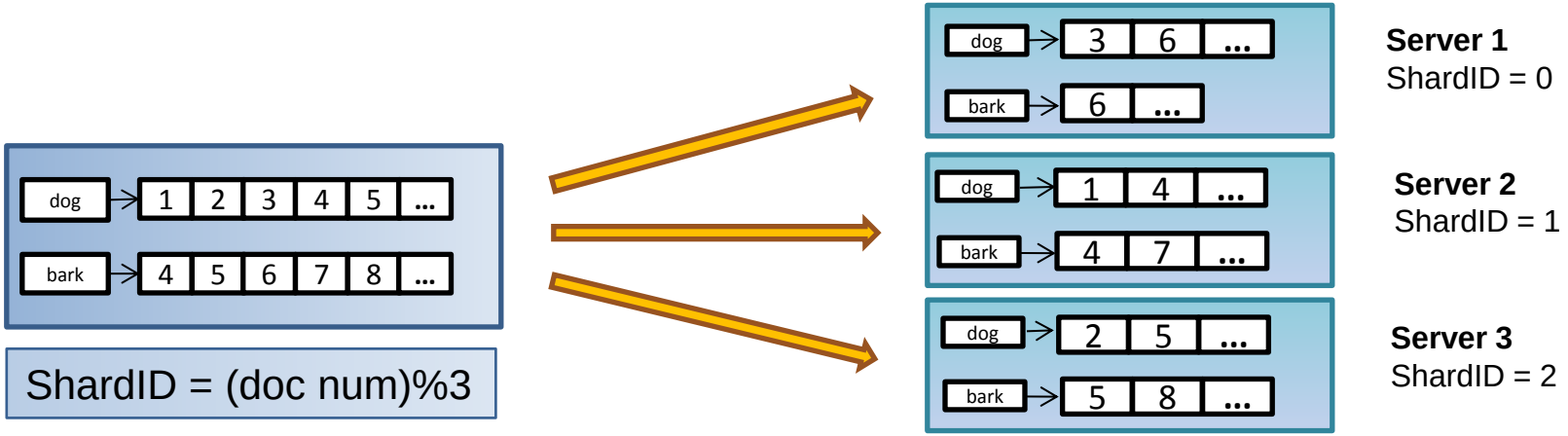


Adjacency lists of dog and bark

Distributing the Problem

What if the adjacency lists are really large? The word dog could appear in lots of documents!

- Partition Adjacency Lists Based on Document Number
- Each server contains fixed range of documents
- To find common documents, adjacency list intersection is performed on each server



Efficient EntityCentric Query Evaluation

Evaluate the SPARQL query

```
SELECT ?X
WHERE {
    ?X contains “dog” .
    ?X contains “barks” .
}
```

and a broad range of other “entity-centric” queries as efficiently as possible

- Design table so that documents (adjacency lists) are distributed uniformly among servers in cluster
- Provide a method to find intersecting documents for each term on each server
- Provides an approach for solving entity-centric queries entirely on the server to reduce network traffic and distribute the workload

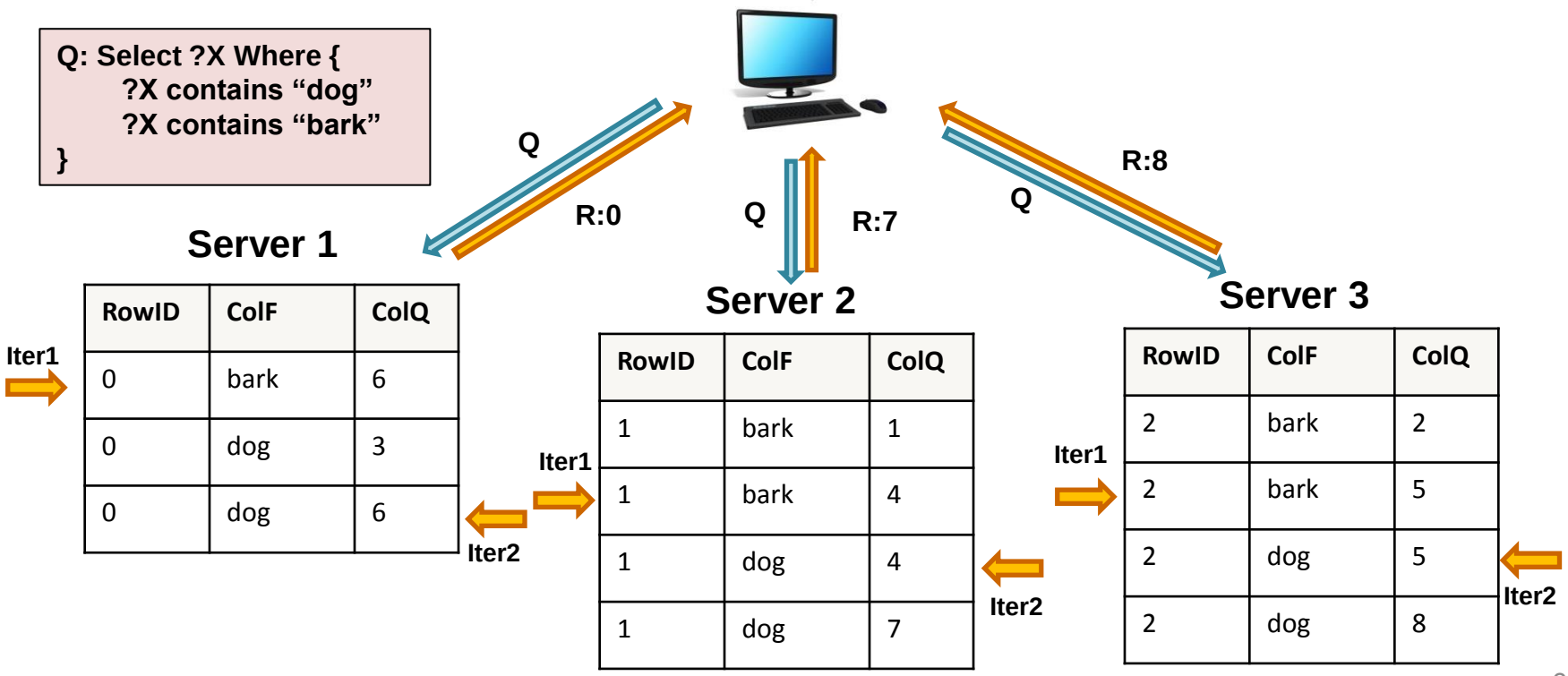
Intersecting Iterators in Accumulo

Elements in adjacency lists of “bark” and “dog” stored in Accumulo in a Document Partitioned Index

- RowID = shardID (doc num % 3)
- Column Family = term (bark or dog)
- Column Qualifier = adjacency element (document number)

Using this index, can evaluate “entity-centric queries” entirely on server

- On each server, iter 1 scans bark and iter 2 scans dog
- Iterators intersect when colQ1 = colQ2, then return result



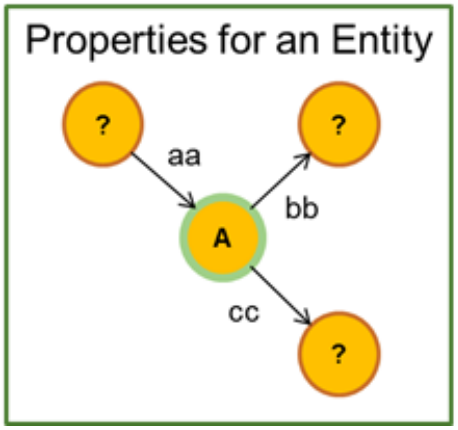
Issuing a query to Doc Partitioned Index

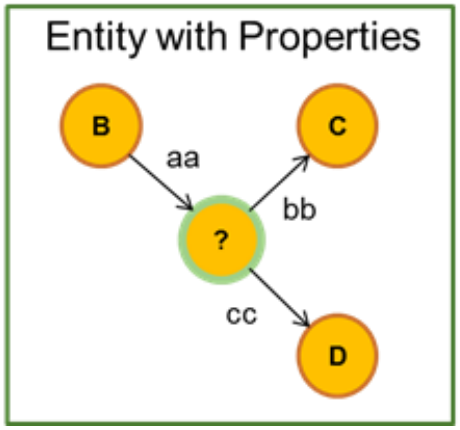
To query a document partitioned index in Accumulo, register intersecting iterator with BatchScanner and set column families

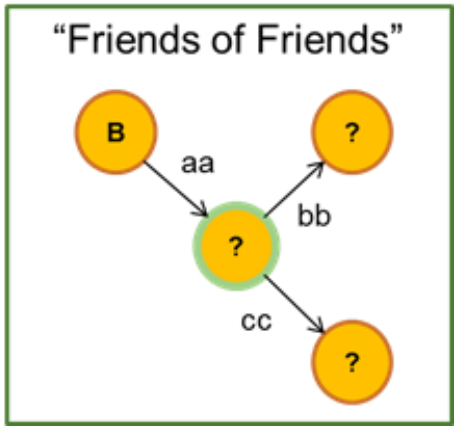
```
Text[] terms = {new Text(“dog"), new Text(“bark")};
BatchScanner bs = conn.createBatchScanner(table, auths, 10);
IteratorSetting is = new IteratorSetting(30, “ii”, IntersectingIterator.class);
IntersectingIterator.setColumnFamilies(is, terms);
bs.addScanIterator(is);
bs.setRanges(Collections.singleton(new Range()));
for(Entry<Key,Value> entry : bs) {
        System.out.println(" " + entry.getKey().getColumnQualifier());
 }
```

Generalizing to a Semantic Network

- Generalize Doc Partitioned Index to accommodate a broad range of SPARQL queries
- Solve as many entity-centric queries server side as possible, where entity centric means all statement patterns share a common variable or constant








```
select ?x ?y ?z
where{
   A aa ?x
   A bb ?y
   A cc ?z
 }
```

```
select ?x
where{
   ?x aa C
   ?x bb B
   ?x cc D
 }
```

```
select ?x ?y ?z
where{
   B aa ?x
   ?x bb ?y
   ?x cc ?z
 }
```

Entity Centric Index Key Design

For each triple (subj, pred, obj, context), include the following two entries in the entity-centric index table:

Accumulo Key		
Row:subj	Column	
	CF:pred	CQ:context\x00Pos\x00obj

Accumulo Key		
Row:obj	Column	
	CF:pred	CQ:context\x00Pos\x00subj

The triple (uri:John, uri:worksAt, uri:Parsons, context: parsonEmployees) would be added as the following two rows:

Row: uri: John, CF: uri:worksAt, CQ: parsonsEmployees\x00object\x00uri:Parsons
Row: uri: Parsons, CF: uri:worksAt, CQ: parsonsEmployees\x00subject\x00uri:John

Observations

- Where's the sharding?
- Lot's of data duplication
- By modifying the Accumulo IntersectingIterator class, can answer the following queries server side
  - Entity with properties
  - Friend of a friend
  - Properties of an entity
- Predicates cannot be variables
- Because Predicates are constants, they can be used to define locality groups to greatly boost performance

Using Entity-Centric Index in Rya

- To use the entity-centric index in Rya, add the following to your normal Rya client configuration:

```
conf.set(ConfigUtils.USE_ENTITY, "true");
conf.set(ConfigUtils.ENTITY_TABLENAME, ENTITY_TABLE_NAME);
```

- This configuration creates the entity-centric index and configures the query planner to delegate portions of the query to the entity-centric index.
  - See EntityDirectExample in rya.indexing.example project
- Once table is created, mutations on entity-centric index can be done through Rya client as mutations are performed on normal Rya tables
- Currently bulk ingest is not supported for entity-centric index
  - WARNING: If bulk ingest is used for core Rya tables, the entity centric will be out of sync with the core Rya tables

Entity-Centric Index and Query Planning

- During query planning, statement patterns in query a group according to common variables and common constants
- Those groups which have the highest “priority” are consolidated into an entity-centric index node