CS698F Advanced Data Management

Instructor: Medha Atre

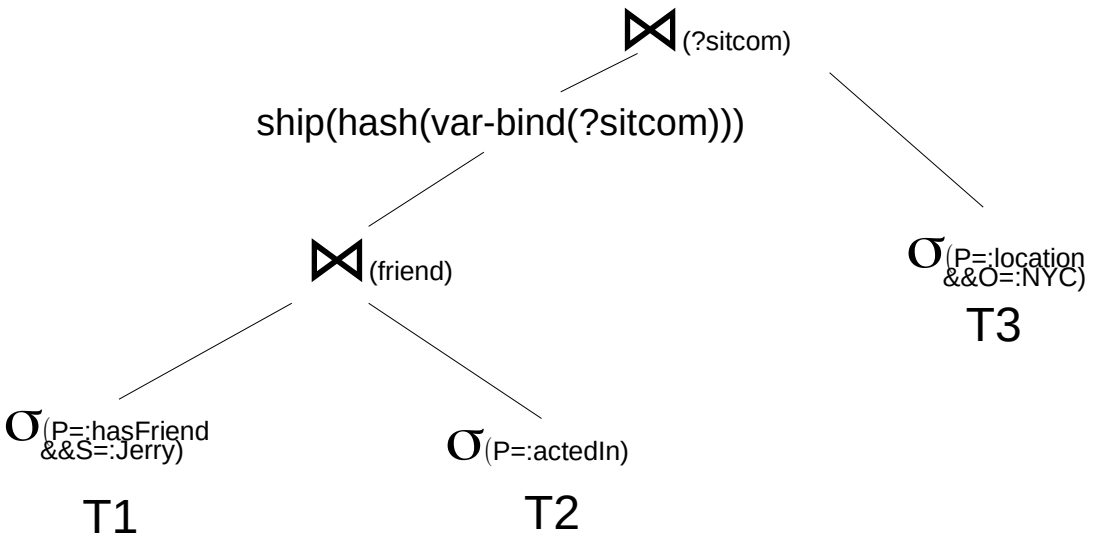
Recap of P2P distribution

- Decide the key-value pair depending on what you "*join*" on.
  - Nodes in case of graph pattern queries.
  - Table columns in case of SQL queries.
- Whenever the join is on the *position* of distribution, it can be done locally
  - If all the joins are on S position with data distributed as ⟨hash(S), list((P,O))⟩, ⟨hash(O), list((P,S))⟩
- Shipping of intermediate results required whenever two consecutive joins are **not** on the same key!

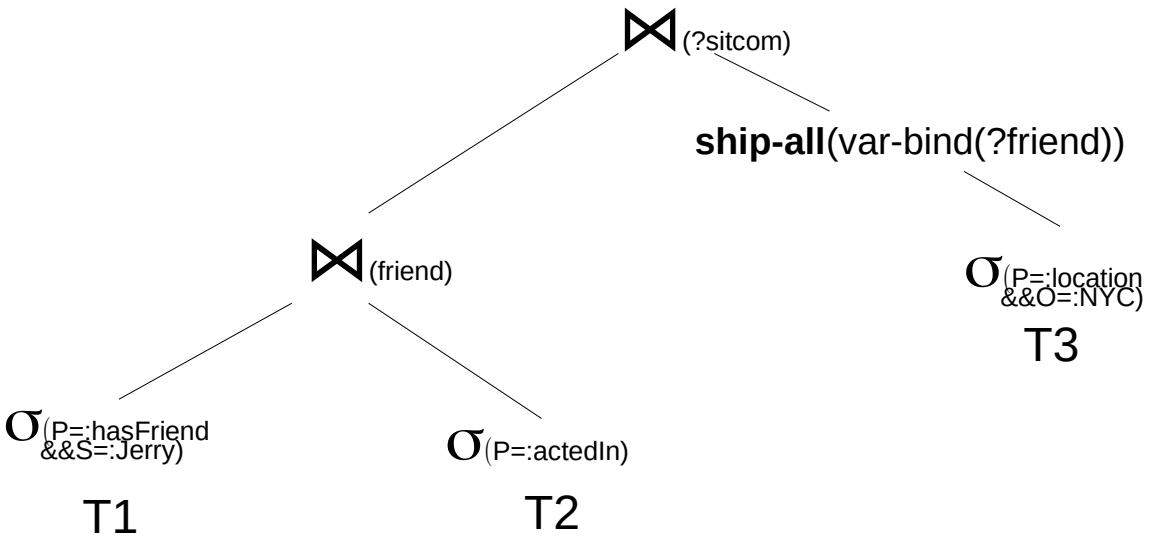
Recap of P2P distribution

- Shipping decision:
  - If the successive join variable exists in the current partial results, reship the partial result by hashing on the binding value of the successive join variable
  - E.g., ⟨*:Jerry, :hasFriend, var-bind(?friend), :actedIn var-bind(?sitcom)*⟩
  - If the successive join variable does **not** exist in the current partial results, **ship-all** intermediate results to all the compute nodes.
  - E.g., Ship each graph edge matching *(?sitcom :location :NYC)* to **ALL** the compute-nodes

Alternate way (bushy)



Alternate way (bushy)



Map-Reduce 101

- Two fundamental functions
  - Map (key1, value1) => (key2, value2)
  - Reduce (key2, list(values2)) => list(key3, values3)
- Mapper takes a list of key-value pairs (depending on what job you define)
- Does some actions, and emits a *different* set of key-value pairs (you can have an *identity* mapper, where key1-value1 = key2-value2
- Sorting/shuffling in between, that combines all the key2-value2 pairs and sends to reducer

Map-Reduce 101

- Reducer takes key-list(value), does some operations and emits values.
- This is a *hierarchical* distributed topology
  - One master, which does sorting/shuffling
  - Several slaves, which do map and reduce jobs.
- Map and Reduce "key" and "values" depend on the "input data" class (Java class) and objects defined in it.
  - Simplest class is "text file".

Word-count example

```
Map (Key dname, Value dcontent)
{
    docfile = open(doc-name)
    for (each line in doc-file) {
        parse words
        create local count
    }
    for (each local word count) {
        emit (word, count);
    }
}
```

```
Reduce (Key word, list(counts))
{
    total = 0;
    for (each count in counts) {
        total += count;
    }
    emit(word, total);
}
```

Word-to-doc index

```
Map (Key dname, Value dcontent)
{
    docfile = open(doc-name)
    for (each line in doc-file) {
        parse words
        emit(word, dname);
    }
}
```

```
Reduce (Key word, list(dname))
{
    //identify reducer
    emit(word, list(dname));
}
```

Distributed grep

```
Map (Key dname, Value dcontent)
{
    docfile = open(doc-name)
    for (each line in doc-file) {
        bool match = grep(regex, line);
        if (match)
            emit(line, dname);
    }
}
```

```
Reduce (Key line, list(dname)) {
    //identify reducer
    emit(line, list(dname));
}
```

Data Distribution

- Simplest way is using standard HDFS commands
  - fs -put <path/to/file>
  - fs -get filename
- Before calling mapper
  - Setup input format – that lets the mapper get correct key-value pairs
  - Setup output format – that lets the reducer write to the correct location

Data Distribution

- Setup sophisticated data distribution using custom data-classes (in Java or other lang).
- Immitate P2P like distribution strategies
- Mapper will get the key-value pairs based on underlying data distribution strategies.
  - Simplest is text files
  - Need additional mapper-reducer jobs for data redistribution.

Some examples/tutorials

- https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm
- https://dzone.com/articles/word-count-hello-word-program-in-mapreduce
- https://research.google.com/archive/mapreduce.html

Joins with Map-Reduce

- Mapper acts as a data distributor
- Reducer acts as a join
- For multiple joins in the same query, iteratively run mapreduce jobs
  - Output of a *previous* map-reduce batch serves as an input to the next map-reduce batch.
  - # of map-reduce batches = # of joins in the query.