MapReduce Design Patterns

Barry Brumitt
barryb@google.com
Software Engineer
About your speaker...

**Ph.D. Robotics, Carnegie Mellon, ‘91 - ’97**
Path Planning for Multiple Mobile Robots

**Researcher, Microsoft Research, ‘98 - ’02**
Ubiquitous Computing

**Software Eng., Microsoft Games, ‘03 - ’05**
AI for Forza Motorsport

**Software Engineer, Google, ‘05 - now**
Maps: Pathfinder
Systems: Infrastructure
MY ROAD TRIP WITH MY BROTHER RAN INTO TROUBLE AROUND PAGE THREE OF THE GOOGLE MAPS PRINTOUT.

<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>Slight Left at RT-22</td>
<td>6.8 mi</td>
</tr>
<tr>
<td>71</td>
<td>Turn Right to stay on RT-22</td>
<td>2.6 mi</td>
</tr>
<tr>
<td>72</td>
<td>Turn Left at Lake Shore Rd</td>
<td>312 ft</td>
</tr>
<tr>
<td>73</td>
<td>Turn Right at Dock St</td>
<td>427 ft</td>
</tr>
<tr>
<td>74</td>
<td>Take the Ferry across the Lake</td>
<td>2.8 mi</td>
</tr>
</tbody>
</table>

OKAY, NOW TAKE DOCK ST TOWARD THE FERRY. WE'RE SUPPOSED TO TAKE A FERRY? IT'S PAST MIDNIGHT, AND THESE WOODS ARE CREEPY.

Google Maps wouldn't steer us wrong.
<table>
<thead>
<tr>
<th>Step</th>
<th>Instruction</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>74.</td>
<td><strong>Take the ferry across the lake.</strong></td>
<td>2.8 mi</td>
</tr>
<tr>
<td>75.</td>
<td>Climb the hill toward Hangman's Ridge, avoiding any mountain lions.</td>
<td>1.172 ft</td>
</tr>
<tr>
<td>76.</td>
<td>When you reach an <strong>old barn</strong>, go around back, knock on the second door, and ask for Charlie.</td>
<td>52 ft</td>
</tr>
<tr>
<td>77.</td>
<td>Tell Charlie the dancing stones are restless. He will give you his van.</td>
<td>Careful</td>
</tr>
<tr>
<td>78.</td>
<td>Take Charley's van down Old Mine Road. Do not wake the straw man.</td>
<td>11 mi</td>
</tr>
<tr>
<td>79.</td>
<td>Turn left on Comstock. When you feel the blood chill in your veins, stop the van and get out.</td>
<td>3.2 mi</td>
</tr>
<tr>
<td>80.</td>
<td>Stand very still. Exits are north, south, and east, but are blocked by a <strong>spectral wolf</strong>.</td>
<td>0 ft</td>
</tr>
<tr>
<td>81.</td>
<td>The spectral wolf fears only fire. The Google Maps team can no longer help you, but if you master the wolf, he will guide you. <strong>GODSPEED.</strong></td>
<td>?? mi</td>
</tr>
</tbody>
</table>
Indexing Large Datasets

All web pages → Index Files → Data Center
Indexing Large Datasets

Geographic Data → Index Files → Data Center

...not so useful for user-facing applications...
Pointer Following (or) Joining

**Feature List**
1: <type=Road>, <intersections=(3)>, <geom>, ...
2: <type=Road>, <intersections=(3)>, <geom>, ...
3: <type=Intersection>, stop_type, POI? ...
4: <type=Road>, <intersections=(6)>, <geom>, ...
5: <type=Road>, <intersections=(3,6)>, <geom>, ...
6: <type=Intersection>, stop_type, POI?, ...
7: <type=Road>, <intersections=(6)>, <geom>, ...
8: <type=Town>, <name>, <geom>, ...

**Intersection List**
3: <type=Intersection>, stop_type, <roads=(
    1: <type=Road>, <geom>, <name>, ...
    2: <type=Road>, <geom>, <name>, ...
    5: <type=Road>, <geom>, <name>, ...
), ...
6: <type=Intersection>, stop_type, <roads=(
    4: <type=Road>, <geom>, <name>, ...
    5: <type=Road>, <geom>, <name>, ...
    7: <type=Road>, <geom>, <name>, ...
), ...

Inner Join Pattern

**Input**
- Feature list

**Map**
- Apply `map()` to each;
- Key = intersection id
- Value = feature

**Shuffle**
- Sort by key

**Reduce**
- Apply `reduce()` to list of pairs with same key, gather into a feature

**Output**
- Feature list, aggregated

---

1: Road
2: Road
3: Intersection
4: Road
5: Road
6: Intersection
7: Road

---

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1: Road</td>
</tr>
<tr>
<td>3</td>
<td>2: Road</td>
</tr>
<tr>
<td>3</td>
<td>3: Intxn</td>
</tr>
<tr>
<td>6</td>
<td>4: Road</td>
</tr>
<tr>
<td>3</td>
<td>5: Road</td>
</tr>
<tr>
<td>6</td>
<td>5: Road</td>
</tr>
<tr>
<td>6</td>
<td>6: Intxn</td>
</tr>
<tr>
<td>6</td>
<td>7: Road</td>
</tr>
</tbody>
</table>

1 3 7 2 5 6

---

3: Intersection
1: Road,
2: Road,
5: Road

6: Intersection
4: Road,
5: Road,
7: Road
**Inner Join Pattern in SQL**

```
SELECT roads.R, roads.D, ints.D
FROM roads INNER JOIN ints
ON roads.I = ints.I
```

**Example Tables:**

<table>
<thead>
<tr>
<th>roads</th>
<th>ints</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>I</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
</tr>
<tr>
<td>6</td>
<td>y</td>
</tr>
<tr>
<td>5</td>
<td>d</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Inner Join Pattern in SQL

```
SELECT roads.R, roads.D, ints.D
FROM roads, ints
WHERE roads.I = ints.I
```

(aka “an Equi Join”)
Tables vs. Flat File?

Tables

- Roads
- Intersections
- Towns

Flat File

- Features
  - Road
  - Intersection
  - Town

Message GeoFeature {
  enum Type {
    ROAD = 1;
    INTERSECTION = 2;
    TOWN = 3;
  }
  required Type type = 0;
  optional Road road = 1;
  optional Intersection intersection = 2;
  optional Town town = 3;
}

“Protocol Buffer”
## References vs. Duplication?

<table>
<thead>
<tr>
<th>References</th>
<th>Duplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: &lt;type=Road&gt;, &lt;intersections=(3)&gt;, &lt;geom&gt;, ...</td>
<td>3: &lt;type=Intersection&gt;, &lt;roads=(1,2,5)&gt;, ...</td>
</tr>
<tr>
<td>2: &lt;type=Road&gt;, &lt;intersections=(3)&gt;, &lt;geom&gt;, ...</td>
<td>1: &lt;type=Road&gt;, &lt;geom&gt;, &lt;name&gt;, ...</td>
</tr>
<tr>
<td>3: &lt;type=Intersection&gt;, &lt;roads=(1,2,5)&gt;, ...</td>
<td>2: &lt;type=Road&gt;, &lt;geom&gt;, &lt;name&gt;, ...</td>
</tr>
<tr>
<td>4: &lt;type=Road&gt;, &lt;intersections=(6)&gt;, &lt;geom&gt;, ...</td>
<td>5: &lt;type=Road&gt;, &lt;geom&gt;, &lt;name&gt;, ...</td>
</tr>
<tr>
<td>5: &lt;type=Road&gt;, &lt;intersections=(3,6)&gt;, &lt;geom&gt;, ...</td>
<td>6: &lt;type=Intersection&gt;, &lt;roads=(5,6,7)&gt;, ...</td>
</tr>
<tr>
<td>6: &lt;type=Intersection&gt;, &lt;roads=(5,6,7)&gt;, ...</td>
<td>4: &lt;type=Road&gt;, &lt;geom&gt;, &lt;name&gt;, ...</td>
</tr>
<tr>
<td>7: &lt;type=Road&gt;, &lt;intersections=(6)&gt;, &lt;geom&gt;, ...</td>
<td>5: &lt;type=Road&gt;, &lt;geom&gt;, &lt;name&gt;, ...</td>
</tr>
<tr>
<td>8: &lt;type=Town&gt;, &lt;name&gt;, &lt;geom&gt;, ...</td>
<td>7: &lt;type=Road&gt;, &lt;geom&gt;, &lt;name&gt;, ...</td>
</tr>
</tbody>
</table>

- **References**: Common primary key; easy restructuring
- **Duplication**: Avoids additional MR passes; denormalizes data
- ...an engineering space / time / complexity tradeoff
class IntersectionAssemblerMapper : public Mapper {
    virtual void Map(MapInput* input) {
        GeoFeature feature;
        feature.FromMapInput(input);
        if (feature.type() == INTERSECTION) {
            Emit(feature.id(), input);
        } else if (feature.type() == ROAD) {
            Emit(feature.intersection_id(0), input);
            Emit(feature.intersection_id(1), input);
        }
    }
};
REGISTER_MAPPER(IntersectionAssemblerMapper);

class IntersectionAssemblerReducer : public Reducer {
    virtual void Reduce(ReduceInput* input) {
        GeoFeature feature;
        GraphIntersection intersection;
        intersection.id = input->key();
        while (!input->done()) {
            feature.FromMapInput(input->value());
            if (feature.type() == INTERSECTION) {
                intersection.SetIntersection(feature);
            } else {
                intersection.AddRoadFeature(feature);
            }
            input->next();
        }
        Emit(intersection);
    }
};
REGISTER_REDUCEINTERSECTIONALINTERSECTIONREDUCER(IntersectionAssemblerReducer);
Join, but no pointers or keys?

<table>
<thead>
<tr>
<th>Input</th>
<th>Map</th>
<th>Shuffle</th>
<th>Reduce</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of items</td>
<td>Apply \texttt{map()} to each; emit (key,val) pairs</td>
<td>Sort by key</td>
<td>Apply \texttt{reduce()} to list of pairs with same key</td>
<td>New list of items</td>
</tr>
</tbody>
</table>

1: Road  
2: Road  
3: Town  
4: Road  
5: Road  
6: Town  
7: Road  

1: Road  
2: Road  
3: Town  
4: Road  
5: Road  
6: Town  
7: Road  

Apply \texttt{map()} to each; emit (key,val) pairs

Sort by key

Apply \texttt{reduce()} to list of pairs with same key

New list of items

1: Road  
2: Road  
3: Town  
4: Road  
5: Road  
6: Town  
7: Road  

3: 1,2,5  
6: 4,5,7
### Bucketing (or) Grace Hash Join

#### Input
- Feature List
  - 1: Road
  - 2: Road
  - 3: Town
  - 4: Road
  - 5: Road
  - 6: Town
  - 7: Road

#### Map
- Emit (key, item) pair
  - Key = geometric hash
  - Secondary key = Type

#### Shuffle
- Sort by keys

#### Reduce
- Intersect all towns with all roads; emit intersecting pairs

#### Output
- (town, road) pair

#### Feature List

<table>
<thead>
<tr>
<th>Feature</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Road</td>
<td>1</td>
</tr>
<tr>
<td>Road</td>
<td>2</td>
</tr>
<tr>
<td>Road</td>
<td>4</td>
</tr>
<tr>
<td>Road</td>
<td>5</td>
</tr>
<tr>
<td>Road</td>
<td>6</td>
</tr>
<tr>
<td>Road</td>
<td>7</td>
</tr>
<tr>
<td>Town</td>
<td>3</td>
</tr>
<tr>
<td>Town</td>
<td>5</td>
</tr>
<tr>
<td>Town</td>
<td>6</td>
</tr>
<tr>
<td>Town</td>
<td>7</td>
</tr>
</tbody>
</table>

#### Diagram

- A: Road, 1
- C: Road, 1
- A: Town, 3
- D: Road, 4
- C: Road, 5
- D: Road, 5
- B: Town, 6
- D: Town, 6
- B: Road, 7
- D: Road, 7

Emit (key, item) pair
- Key = geometric hash
- Secondary key = Type

Sort by keys

Intersect all towns with all roads; emit intersecting pairs

A B
C D
Reduce on Key A

<table>
<thead>
<tr>
<th>Input</th>
<th>Map</th>
<th>Shuffle</th>
<th>Reduce</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature List</td>
<td>Emit (key, item)</td>
<td>Sort by</td>
<td>Intersect all towns with</td>
<td>(town, road) pair</td>
</tr>
<tr>
<td></td>
<td>pair</td>
<td>keys</td>
<td>all roads; emit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Key = geometric</td>
<td></td>
<td>intersecting pairs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hash</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Secondary key =</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature List</th>
<th>Emit (key, item)</th>
<th>Sort by</th>
<th>Intersect all towns with</th>
<th>(town, road) pair</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Road</td>
<td>(A-Road, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2: Road</td>
<td>(C-Road, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3: Town</td>
<td>(C-Road, 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4: Road</td>
<td>(A-Town, 3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5: Road</td>
<td>(B-Town, 3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6: Town</td>
<td>(C-Town, 3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7: Road</td>
<td>(D-Road, 4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(C-Road, 5)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(D-Road, 5)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(B-Town, 6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(D-Town, 6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(B-Road, 7)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(D-Road, 7)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(A-Town, 3)
(A-Road, 1)

(3, 1)
### Reduce on Key B

<table>
<thead>
<tr>
<th>Input</th>
<th>Map</th>
<th>Shuffle</th>
<th>Reduce</th>
<th>Output</th>
</tr>
</thead>
</table>
| Feature List | Emit (key, item) pair  
Key = geometric hash  
Secondary key = Type | Sort by keys | Intersect all towns with all roads; emit intersecting pairs | (town, road) pair |

#### Feature List

- 1: Road
- 2: Road
- 3: Town
- 4: Road
- 5: Road
- 6: Town
- 7: Road

#### Map

- (A-Road, 1)
- (C-Road, 1)
- (C-Road, 2)
- (A-Town, 3)
- (B-Town, 3)
- (C-Town, 3)
- (D-Road, 4)
- (C-Road, 5)
- (D-Road, 5)
- (B-Town, 6)
- (D-Town, 6)
- (B-Road, 7)
- (D-Road, 7)

#### Diagram

- Intersecting pairs:
  - (B-Town, 3)
  - (B-Town, 6)
  - (B-Road, 7)

- Output:
  - (6, 7)
Reduce on Key C

**Input**
- Feature List

**Map**
- Emit (key, item) pair
  - Key = geometric hash
  - Secondary key = Type

**Shuffle**
- Sort by keys

**Reduce**
- Intersect all towns with all roads; emit intersecting pairs

**Output**
- (town, road) pair

### Feature List
- 1: Road
- 2: Road
- 3: Town
- 4: Road
- 5: Road
- 6: Town
- 7: Road

### Map
- (A-Road, 1)
- (C-Road, 1)
- (C-Road, 2)
- (A-Town, 3)
- (B-Town, 3)
- (C-Town, 3)
- (D-Road, 4)
- (C-Road, 5)
- (D-Road, 5)
- (B-Town, 6)
- (D-Town, 6)
- (B-Road, 7)
- (D-Road, 7)

### Shuffle
- (C-Town, 3)
- (C-Road, 1)
- (C-Road, 5)
- (C-Road, 2)

### Output
- (3, 1)
- (3, 2)
- (3, 5)
Reduce on Key D

<table>
<thead>
<tr>
<th>Input</th>
<th>Map</th>
<th>Shuffle</th>
<th>Reduce</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature List</td>
<td>Emit (key, item) pair</td>
<td>Sort by keys</td>
<td>Intersect all towns with all roads; emit intersecting pairs</td>
<td>(town, road) pair</td>
</tr>
<tr>
<td></td>
<td>Key = geometric hash</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Secondary key = Type</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1: Road  (A-Road, 1)
2: Road   (C-Road, 1)
3: Town   (A-Town, 3)
4: Road   (B-Town, 3)
5: Road   (C-Town, 3)
6: Town   (D-Road, 4)
7: Road   (C-Road, 5)
5: Road   (D-Road, 5)
6: Town   (B-Town, 6)
6: Town   (D-Town, 6)
7: Road   (B-Road, 7)
7: Road   (D-Road, 7)

1: Road  (A-Road, 1)
2: Road   (C-Road, 1)
3: Town   (A-Town, 3)
4: Road   (B-Town, 3)
5: Road   (C-Town, 3)
6: Town   (D-Road, 4)
5: Road   (D-Road, 5)
6: Town   (D-Town, 6)
7: Road   (D-Road, 7)

Emit (key, item) pair
Key = geometric hash
Secondary key = Type
Sort by keys
Intersect all towns with all roads; emit intersecting pairs
(town, road) pair
Output… not quite…

<table>
<thead>
<tr>
<th>Input</th>
<th>Map</th>
<th>Shuffle</th>
<th>Reduce</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature List</td>
<td>Apply \texttt{map()} to each; emit (key,val) pairs</td>
<td>Sort by key</td>
<td>Apply \texttt{reduce()} to list of pairs with same key</td>
<td>List of items</td>
</tr>
</tbody>
</table>

1: Road
2: Road
3: Town
4: Road
5: Road
6: Town
7: Road

(3, 1)
(6, 7)
(3, 1)
(3, 2)
(3, 5)
(6, 7)
(6, 4)
(6, 5)
...recall earlier Join Pattern

**Input**: List of items

**Map**: Apply `map()` to each; emit (key, val) pairs

**Shuffle**: Sort by key

**Reduce**: Apply `reduce()` to list of pairs with same key

**Output**: New list of items

<table>
<thead>
<tr>
<th>Input</th>
<th>Map</th>
<th>Shuffle</th>
<th>Reduce</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Road</td>
<td>(3, 1: Road)</td>
<td></td>
<td>(3, 1: Road)</td>
<td>3: Intersection</td>
</tr>
<tr>
<td>2: Road</td>
<td>(3, 2: Road)</td>
<td></td>
<td>(3, 2: Road)</td>
<td>1: Road, 2: Road</td>
</tr>
<tr>
<td>3: Intersection</td>
<td>(3, 3: Intxn)</td>
<td></td>
<td>(3, 3: Intxn.)</td>
<td>3: Intersection</td>
</tr>
<tr>
<td>4: Road</td>
<td>(6, 4: Road)</td>
<td></td>
<td>(3, 5: Road)</td>
<td>4: Road, 5: Road</td>
</tr>
<tr>
<td>5: Road</td>
<td>(3, 5: Road)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6: Intersection</td>
<td>(6, 6: Intxn)</td>
<td></td>
<td></td>
<td>6: Intersection</td>
</tr>
<tr>
<td>7: Road</td>
<td>(6, 6: Intxn)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- 1: Road
- 2: Road
- 3: Intersection
- 4: Road
- 5: Road
- 6: Intersection
- 7: Road

Graph:
- 1, 2, 3, 4, 5, 6, 7
- Connections: 1->2, 1->3, 3->4, 3->5, 5->6, 6->7
Recursive Key Join Pattern

<table>
<thead>
<tr>
<th>Input</th>
<th>Map</th>
<th>Shuffle</th>
<th>Reduce</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output from previous phase</td>
<td>Identity Mapper, key = town</td>
<td>Sort by key</td>
<td>Reducer sorts, gathers, remove duplicates; similar to join</td>
<td>Index of roads in each town</td>
</tr>
</tbody>
</table>

Could use 2ndry keys to avoid reduce sort(), eg: 

(6-7, 7)
Chained MapReduce’s Pattern

Input
- Feature List
- (town, road) pair

Map
- Emit (key, item) pair
  - Key = geometric hash
  - Secondary key = Type
- Identity Mapper, key = town

Shuffle
- Sort by keys
- Sort by key

Reduce
- Intersect all towns with all roads; emit intersecting pairs
- Reducer sorts, gathers, remove duplicates; similar to join

Output
- (town, road) pair
- Index of roads in each town
Distributing Costly Computation: e.g. Rendering Map Tiles

Input: Geographic feature list

Map: Emit each to all overlapping latitude-longitude rectangles

Shuffle: Sort by key (key= Rect. Id)

Reduce: Render tile using data for all enclosed features

Output: Rendered tiles

(Bucket pattern) (Parallel rendering)
Finding Nearest Points Of Interest (POIs)

**Feature List**

1. Type, Road, Intersection, ...
2. Type, Road, Intersection, ...
3. Type, Road, Intersection, ...
   ...

**Nearest POI within 5mi of Intersection**

(1, 1)
(2, 1)
(3, 1)
(4, 1)
(5, 7)
(6, 7)
(7, 7)
(8, 7)
(9, 7)
Finding Nearest POI on a Graph

<table>
<thead>
<tr>
<th>Input</th>
<th>Map</th>
<th>Shuffle</th>
<th>Reduce</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>Perform Dijkstra from each POI node. Emit POI &amp; dist. At each node in search.</td>
<td></td>
<td>For each node, emit closest POI</td>
<td>Nodes with nearest POIs</td>
</tr>
</tbody>
</table>

Diagram:
- Red node: Starting point
- Yellow node: Example POI
- Graph: Connections between nodes
Finding Nearest POI on a Graph

**Input**
- Graph

**Map**
- Perform Dijkstra from each POI node.
- Emit POI & dist. At each node in search.

**Shuffle**

**Reduce**
- For each node, emit closest POI

**Output**
- Nodes with nearest POIs
Finding Nearest POI on a Graph

**Input**
- Graph

**Map**
- Perform Dijkstra from each POI node.
- Emit POI & dist. At each node in search.

**Shuffle**

**Reduce**
- For each node, emit closest POI

**Output**
- Nodes with nearest POIs
Finding Nearest POI on a Graph

Input: Graph

Map: Perform Dijkstra from each POI node. Emit POI & dist. At each node in search.

Shuffle: 

Reduce: For each node, emit closest POI

Output: Nodes with nearest POIs
Putting it all together: Nearest POI

**Input** → **Map** → **Shuffle** → **Reduce** → **Output**

*Use key-join pattern to create nodes, edges out of intersections, roads*

*Use bucketing pattern to create “appropriate” (overlapping, large-enough) subgraphs*

*Nodes with edges* → *Subgraphs*

**Subgraphs**

Perform Dijkstra from each POI node. Emit POI & dist. At each node in search.

*For each node, emit closest POI*

*Nodes with nearest POI & dist*

**Nodes with nearest POI & dist** → **Use identity mapper & gather pattern to sort and clean-up node, POI pairs**

*Sorted nodes with nearest POI*
Hard Problems for MapReduce

• Following multiple pointer hops
• Iterative algorithms
• Algorithms with global state
• Operations on graphs without good embeddings

• [insert your favorite challenge here]
Summary

MapReduce eases:
- Machine coordination
- Network communication
- Fault tolerance
- Scaling
- Productivity

MapReduce patterns:
- “Flat” data structures
- Foreign / Recursive Key Joins (aka pointer following)
- Hash Joins (aka bucketing)
- Distribute $$ computation
- Chain MapReduce phases
- Simplify Reduce() by using secondary keys
- [ insert your pattern here ]
Questions?

- **MapReduce:** Simplified Data Processing on Large Clusters,
  Jeffrey Dean and Sanjay Ghemawat
  *OSDI’04: Sixth Symposium on Operating System Design and Implementation*

- **Contact:** barryb@google.com