CSCI4180 Tutorial Week 9
Assignment 2
Parallel BFS Implementation

31 October 2013

Jeremy Chan
SHB118
cwchan@cse.cuhk.edu.hk

Overview of Assignment 2

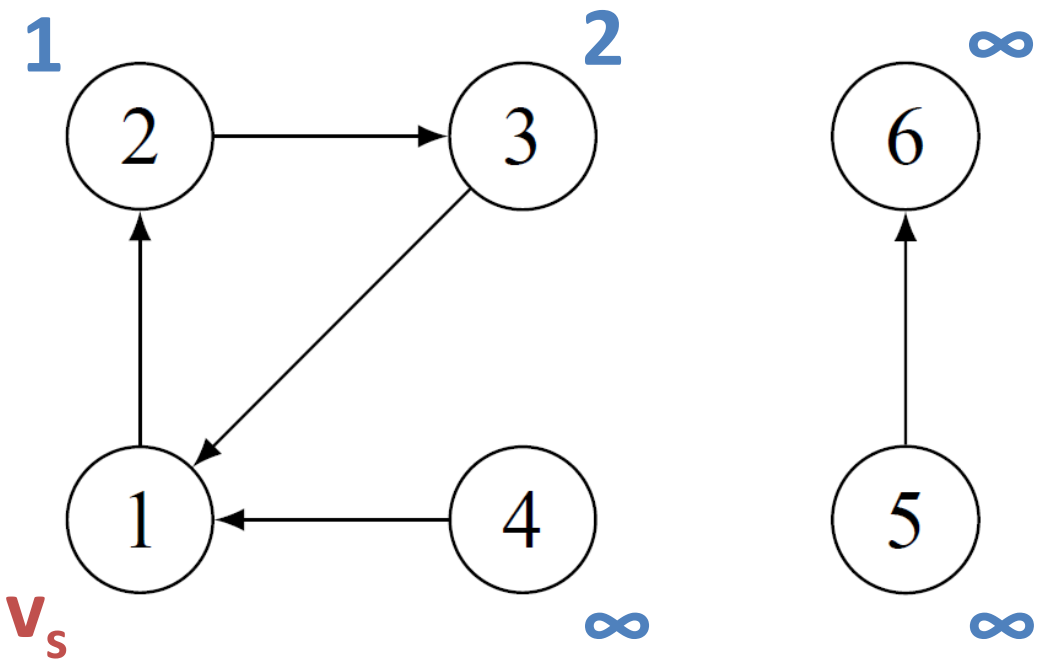
- **Part 1 HBase (50%)**
  - Database-like system on Hadoop
  - Mostly configuration work
  - Will be covered next week in tutorial

- **Part 2 Parallel BFS (50%)**
  - Single-source shortest path lengths
  - Learn to run MapReduce **iteratively**

- **5% Bonus**

Definition

- Given a graph G = (V,E) and a source node $v_s \in V$, find the **shortest path length** (distance) from v_s to every other **reachable** node in V.



First Step – Parsing Input

- From ***Input Format*** to ***adjacency list format***
  - So we make sure each node goes to the same mapper

Input Format

1	3
1	5
3	1
2	4
4	2

Adjacency List

1	3, 5
3	1
2	4
4	2

- You can design a separate Mapper/Reducer to do the transformation

Node Structure

- Three states (Color) of a node:
  - WHITE: Not visited 
  - GRAY: Discovered 
  - BLACK: Processed 

- Stores distance from v_s as internal state

Distance means the shortest path length

- Keeps track of neighbours by adjacency list

Adjacency matrix is difficult for MapReduce

Hints for Designing the Node Class

- Use an `enum` for the `color`

```
public static enum Color {
    WHITE, GRAY, BLACK
};
```

- Use `int` for `id` and `distance`
- Use `List<Integer>` for the adjacency list
- Design a proper constructor
  - Think about the initial `color`, `id` and `distance`

How to Emit a Node?

The Java Way

- Implements a `toString()` method and a `fromString()` method in your Node Class
- Emit a node simply by a **org.apache.hadoop.io.Text**

	IntWritable	Text		
e.g.	`1`	`2,3,4,5`	`0`	BLACK
	NodeID	Adj. List	Distance	Color

- However, functions like `String.split()` could be slow

How to Emit a Node?

The Hadoop Way (Recommended)

- Implements ***Writable*** in the Node Class
  - You will need to provide implementation for the `readFields()` and `write()` method
- Emit a Node as the value directly
- Please refer to Hadoop: The Definitive Guide, Chapter 4 - Hadoop I/O

Challenges in Parallel BFS

```
1: class Mapper
2:     method Map(nid n, node N)
3:         d ← N.Distance
4:         Emit(nid n, N)
5:         for all nodeid m ∈ N.AdjacencyList do
6:             Emit(nid m, d + 1)

1: class Reducer
2:     method Reduce(nid m, [d1, d2, ...])
3:         d_min ← ∞
4:         M ← ∅
5:         for all d ∈ counts [d1, d2, ...] do
6:             if IsNode(d) then
7:                 M ← d
8:             else if d < d_min then
9:                 d_min ← d
10:        M.Distance ← d_min
11:        Emit(nid m, node M)
```

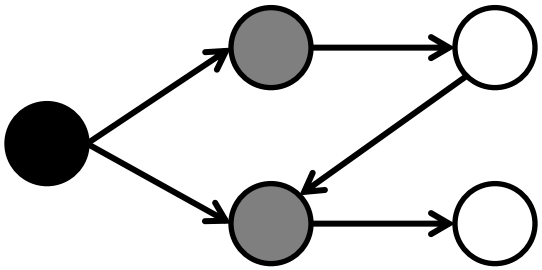
How to process GRAY nodes only?

How to emit two kinds of data?

How to continue to the next iteration?

Parallel BFS in MapReduce

- **MAPPER Design**



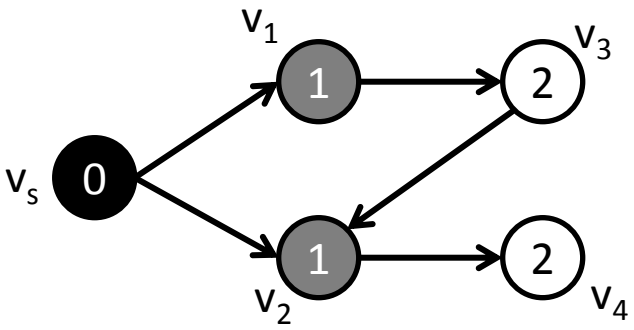
1. **Mappers** should not do anything to **BLACK** or **WHITE** nodes (not their turn to process), just emit the nodes “as-is”

2. **Mappers** processing **GRAY** node should
1) mark its neighbours as **GRAY**,
2) set neighbours’ distance
3) mark itself **BLACK**
4) emits itself and its neighbours

Mapper's Limitation

1. **Mapper** which has v_1 emits v_3 as GRAY, another **Mapper** emits v_3 as WHITE

2. **Mapper** which has v_1 emits v_3.distance = 2, another **Mapper** emits v_3.distance = INF



3. **Mapper** which has v_1 emits v_3 without knowing its adjacency list, another **Mapper** which has v_3 emits v_3 with its adjacency list

Reducer solves consistency problem

Parallel BFS in MapReduce

- **REDUCER Design**

- For a Node, different mappers will provide conflicting information
- Think about how to combine the following three properties
  - Adjacency list
  - Distance
  - Color

- The emitted node can then be used for the Mapper in the next iteration

How to Chain MapReduce Jobs?

- Each "Map + Reduce" only explored one step further from v_s, we need to repeat this process
  - **Map1 → Reduce1 → Map2 → Reduce2 → Map3…**

The Java Way

- Drive jobs in the `main()` function using loops, counter and conditionals
- Different `Configuration` and `Job` Object in each iteration
- Set `job.waitForCompletion(true);`
- Jobs communicates by writing and reading intermediate files on Hadoop DFS
  - **For Job_i**
    - `FileInputFormat.addInputPath (`***OutputPath of*** Job_{i-1}`)`

How to Chain MapReduce Jobs?

The Hadoop Way

- Use **org.apache.hadoop.mapred.jobcontrol**
- In this Hadoop version (0.20.203), use the following deprecated classes to chain jobs

```
mapred.jobControl.Jobcontrol ->
mapred.jobControl.Job ->
mapred.jobConf ->
mapred.Mapper/Reducer/MapReduceBase
```

How to Chain MapReduce Jobs?

- **Add a Job to JobControl**
  - `Job job1 = new Job (jobConf1)`
  - `Job job2 = new job (jobConf2)`
- **Add dependency (e.g. 2 depends on 1)**
  - `job2.addDependingJob(job1)`
- **Add jobs to JobControl**
  - `JobControl jc = new JobControl ();`
  - `jc.addJob(job1);`
  - `jc.addJob(job2);`
- **Run JobControl**
  - `jc.run()`
- **Check if job is finished**
  - `jobControl.allFinished()  // start this in a thread!`

When to Stop?

- Stop when all reachable nodes are processed
- **i.e. When there are no more GRAY nodes**
  - Nodes are either BLACK or WHITE
- How to check?
  - Use **org.apache.hadoop.mapred.Counters**

Hadoop Counter: Usage

1. Declare Counter

```
public static enum GrayCounter {
  COUNT
};
```

2. Increment Counter

```
context.getCounter(GrayCounter.COUNT).increment(1);
```

3. Retrieve Counter Value

```
int grayCount =
job.getCounters().findCounter(ParallelBfs.GrayCounter.COUNT).getValue();
```

If you want to get the correct value of a Counter, only call `getValue()` when the job has finished running

Final Step – Clean-up

- Transform the output of the last run to the required format
- Only outputs the tuple which the node is ***reachable***
- Feel free to use another set of Mapper/Reducer to do this step

Coming Next…

Nov 7

- HBase setup procedures
- How to write MapReduce programs for HBase

Due: Nov 14

- Demo on the next day
  - Only your last submission will be graded
  - Make sure your filenames are correct

Reminders

Tips

- Test your program correctness with hand-craft test cases (loops, directed edge, etc.)
- Running time depends on the number of passes of Map + Reduce
  - Largest test case in the demo will be Twitter_3, can take 5-15 mins to finish

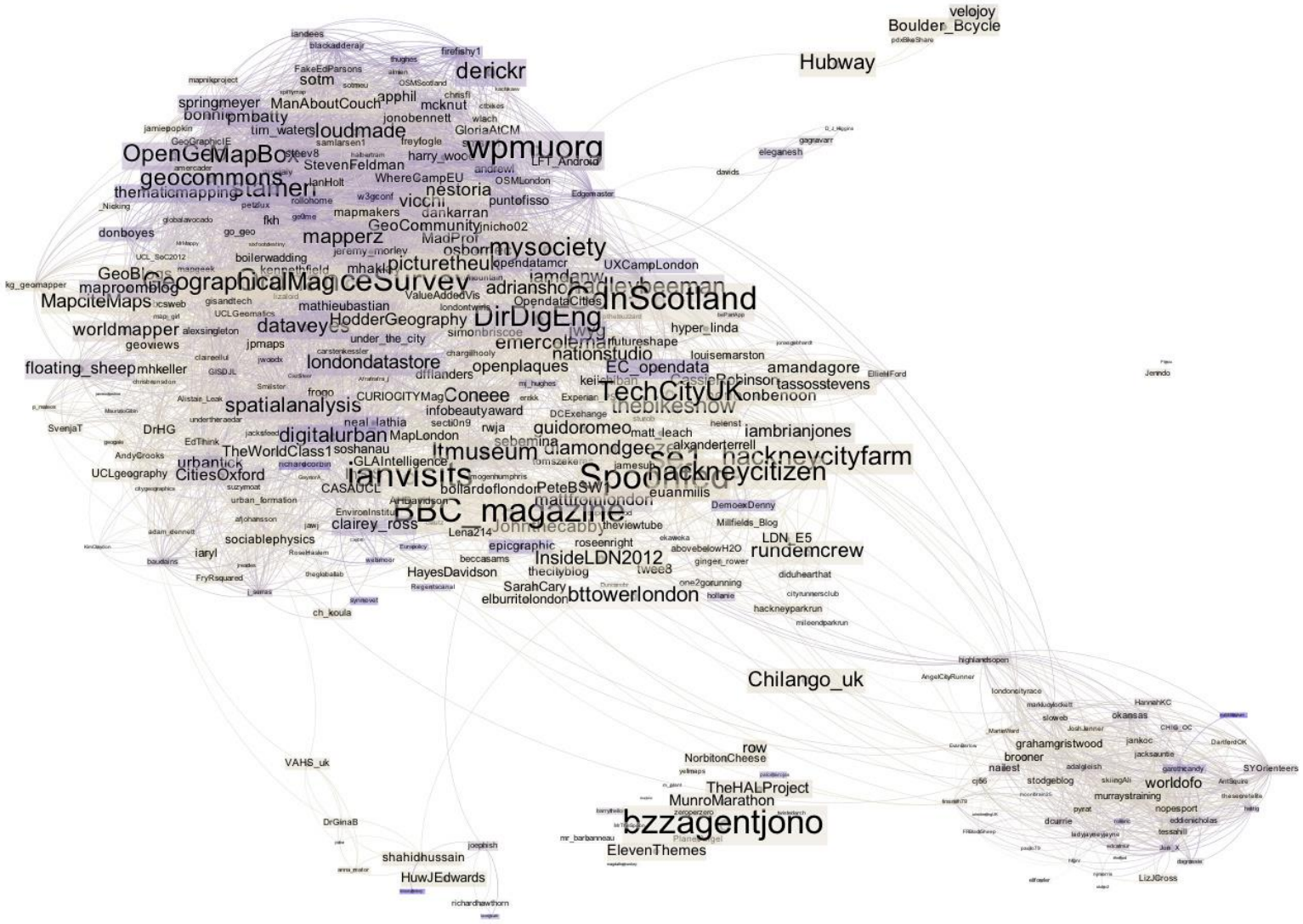
- Should be more challenging than Assignment 1
- Please start early!

Questions?

Thank You

Visualizing Twitter Social Graph

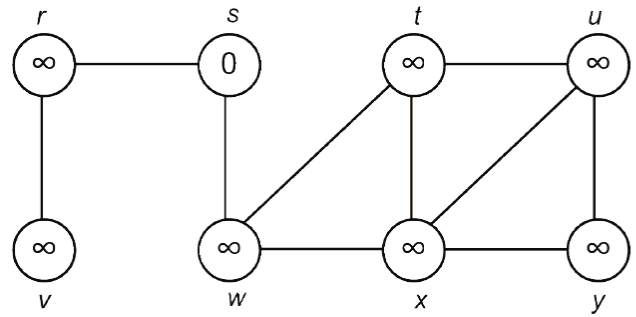
Reference



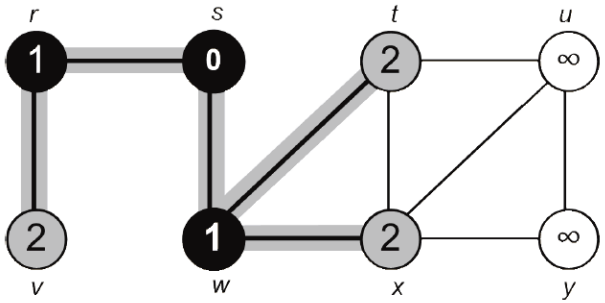
http://oliverobrien.co.uk/2012/07/six-degrees-of-twitter/

BFS: Example

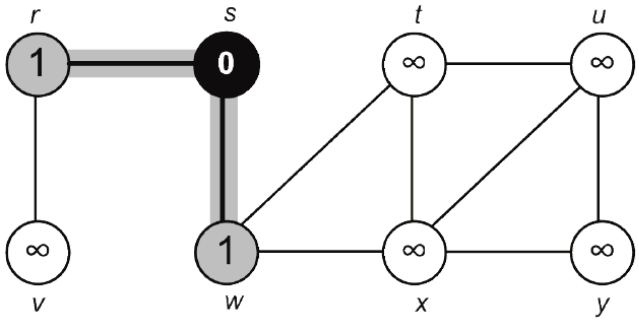
Reference



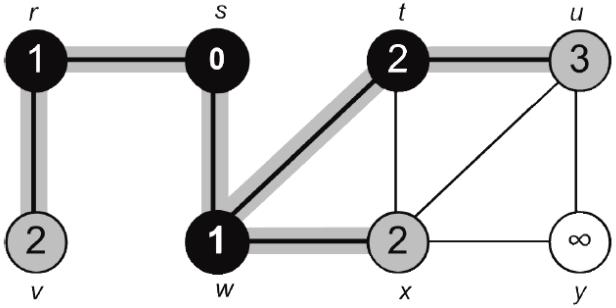
Q | s |
0



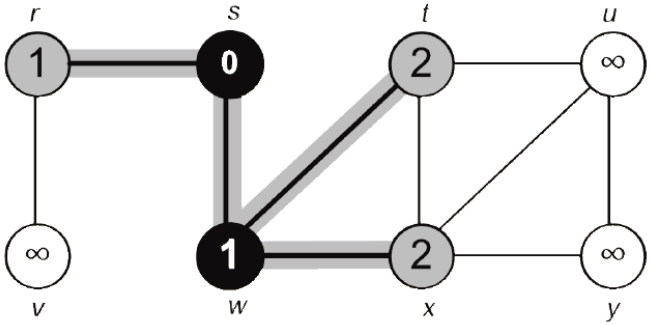
Q | t | x | v |
2 2 2



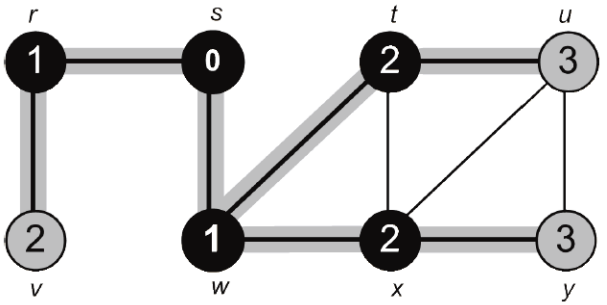
Q | w | r |
1 1



Q | x | v | u |
2 2 3



Q | r | t | x |
1 2 2

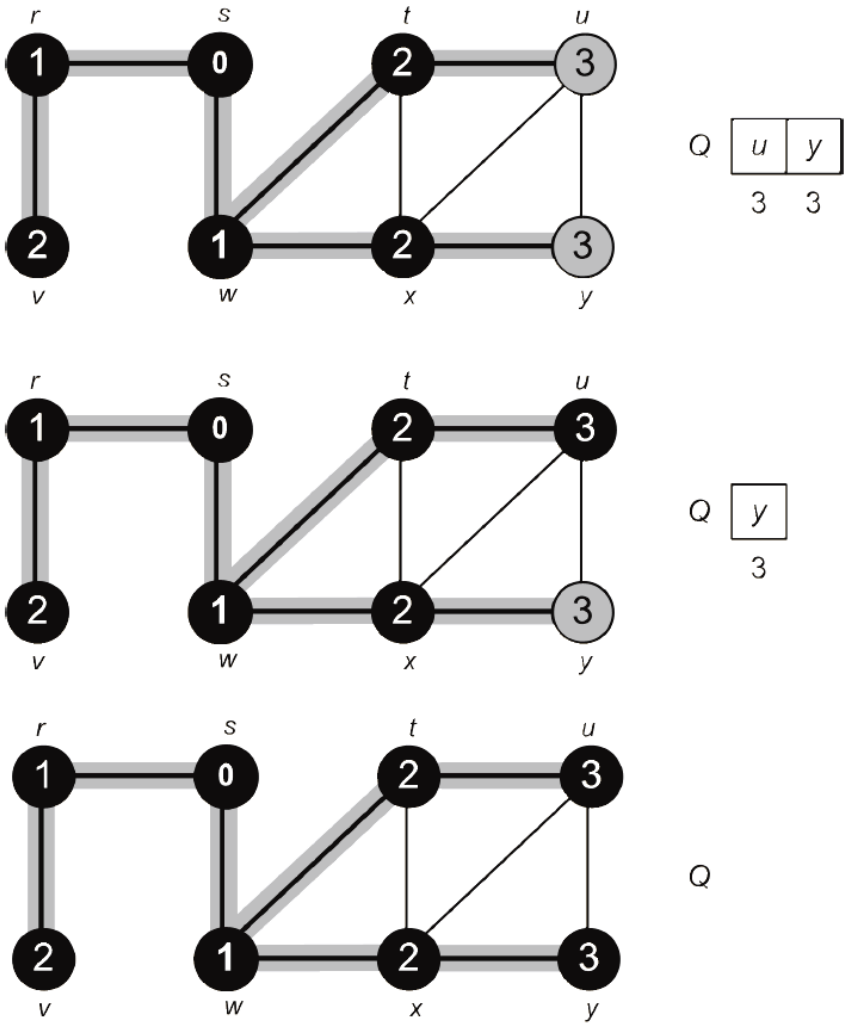


Q | v | u | y |
2 3 3

Credit: Dr. Matthew Tang's Data Structure Notes

BFS: Example

Reference



- For the Assignment, the graph is **directed**
- Some nodes can be unreachable

Credit: Dr. Matthew Tang's Data Structure Notes