ECE 454 Computer Systems Programming

Data Analytics with Map Reduce

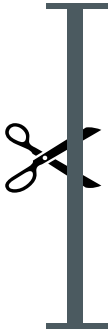
Jon Eyolfson
Courtesy: Ashvin Goel
ECE Dept, University of Toronto

The Course Until Now

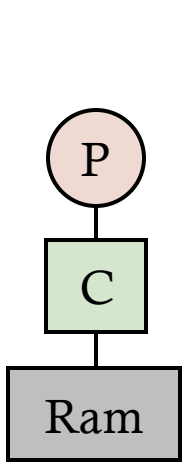
Sequential program optimization:

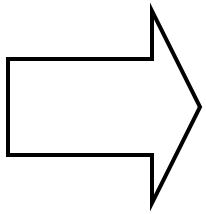
- CPU architecture
- Compiler optimization
- Cache optimization
- Dynamic memory

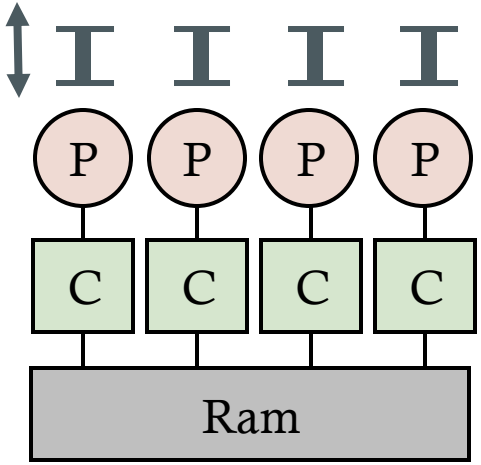
Parallel programming on single machine:



- Threads
- Synchronization
- Parallel architectures
- Better locks
- Avoiding locks







Next Step

Internet-Scale Data

Millions of Terabytes of data.

Google indexed roughly 200 Terabytes, or .004% of the entire internet
-- Eric Schmidt, former Google CEO

1.19 billion active users.

3.5 billion pieces of content shared per day.

10.5 billion minutes spent on Facebook worldwide every day.

Challenges

- Index and analyze data?
- Store data?
- Serve data with low latency?

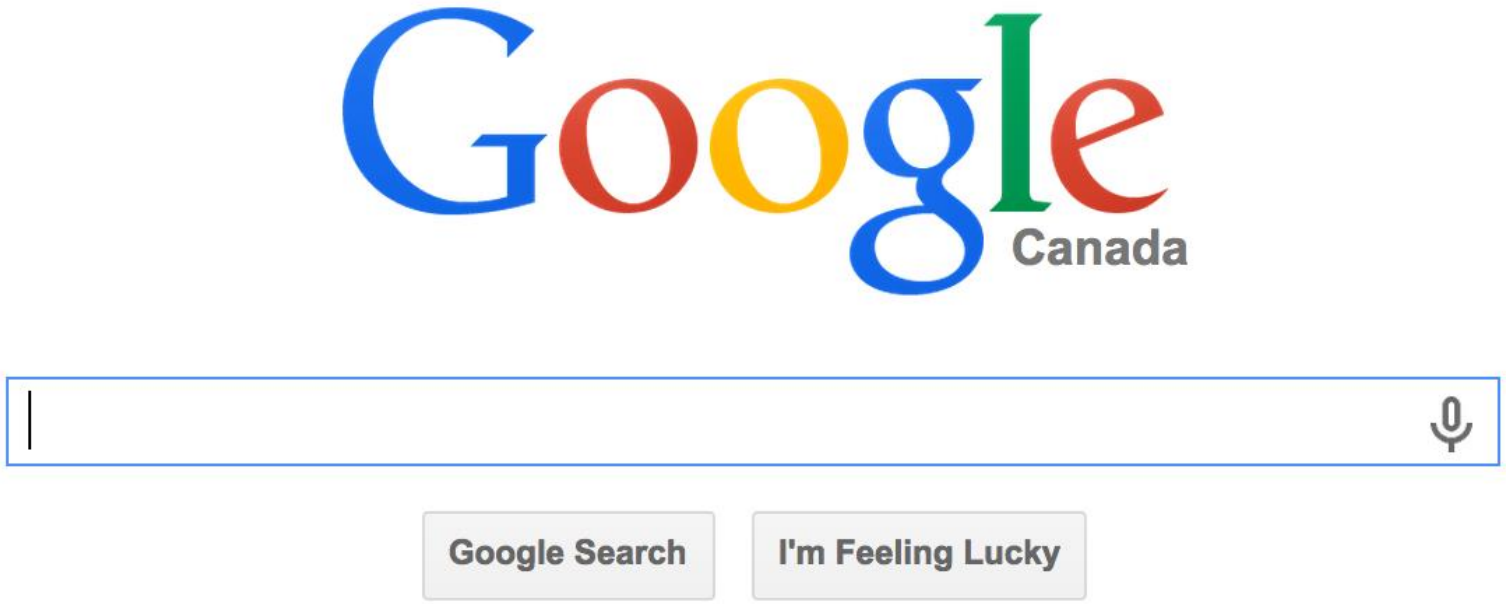
Big Data Analytics: this lecture

How Facebook works: later…

Big Data Analytics

- How can we perform (simple) computations on Internet-scale data?
  - Grep, sort, word-count
  - Index pages, find who references a web pages
  - Log analysis

How do you Build Google?



How Google Works

1. The web server sends the query to the index servers. The content inside the index servers is similar to the index in the back of a book--it tells which pages contain the words that match any particular query term.

2. The query travels to the doc servers, which actually retrieve the stored documents. Snippets are generated to describe each search result.

3. The search results are returned to the user in a fraction of a second.

Rest of the lecture focuses on the index servers

Two Indexing Challenges

- Which webpages contain the given keyword (e.g., "NBA")?
  - Problem is called web page indexing
  - Need to crawl and analyze all web pages
  - Output: <word, list(URLs)>
    - Example: <"NBA", (www.nba.com, www.espn.com, …)>
- Which webpages for the given keyword are important?
  - Problem is called web page ranking
  - Need to first find pages that link to a page
  - Output: <target, list(source)>
    - Example: <www.nba.com, (www.espn.com, www.cnn.com, …)>
  - Need to rank pages based on output (PageRank)

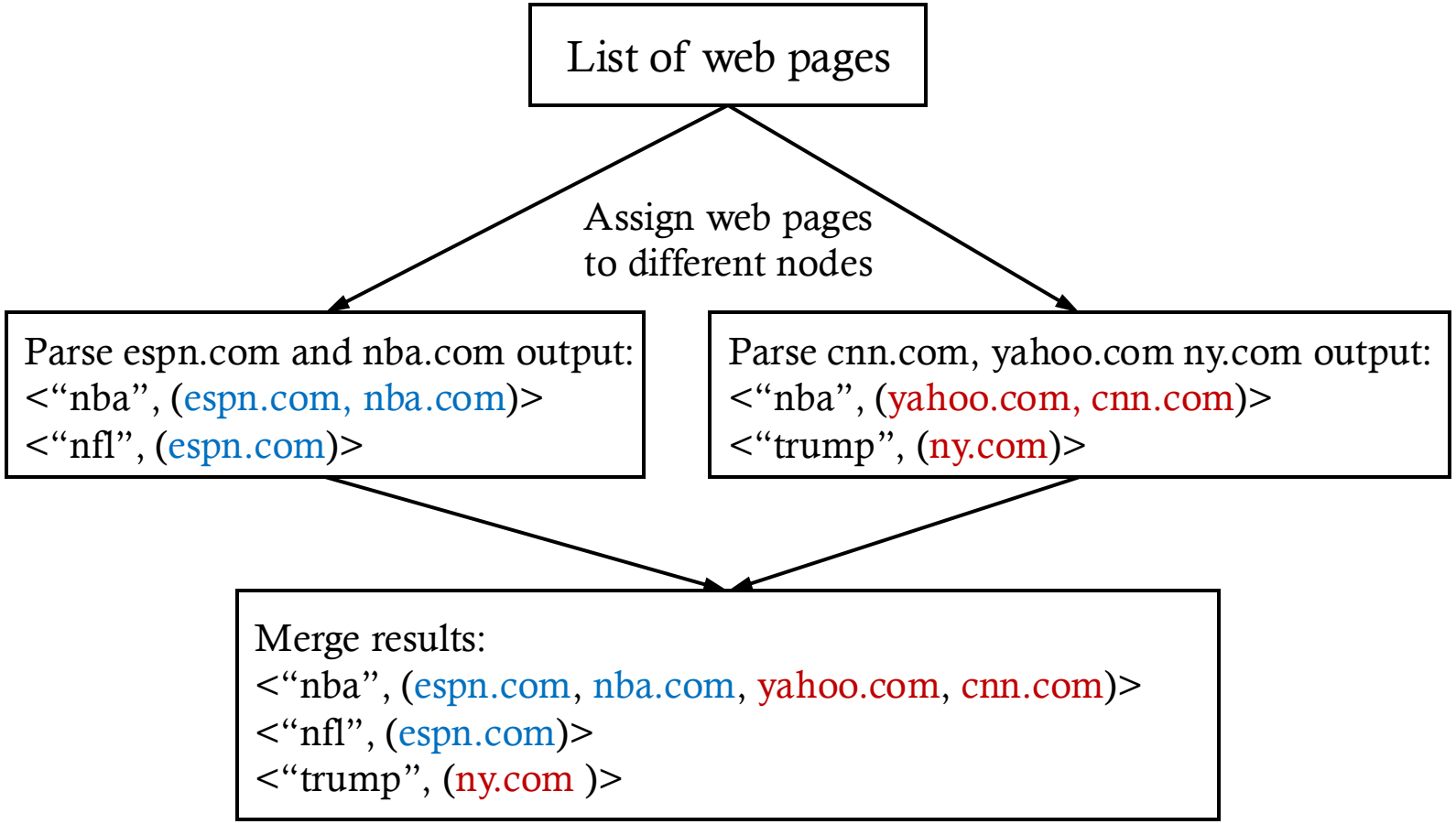
Web Page Indexing

```
// input: list of all web pages
// output: for each word, web pages that contain the word

index(List webpages) {
  Hash output = new Hash<string word, List<string url>>;

  for each page p in webpages {
    for each word w in p {
      if (!output.exists(w))
        output{w} = new List<string>;
      // append web page for this word
      output{w}.push(URL(p));
    }
  }
}
```

What if we have billions of web pages?

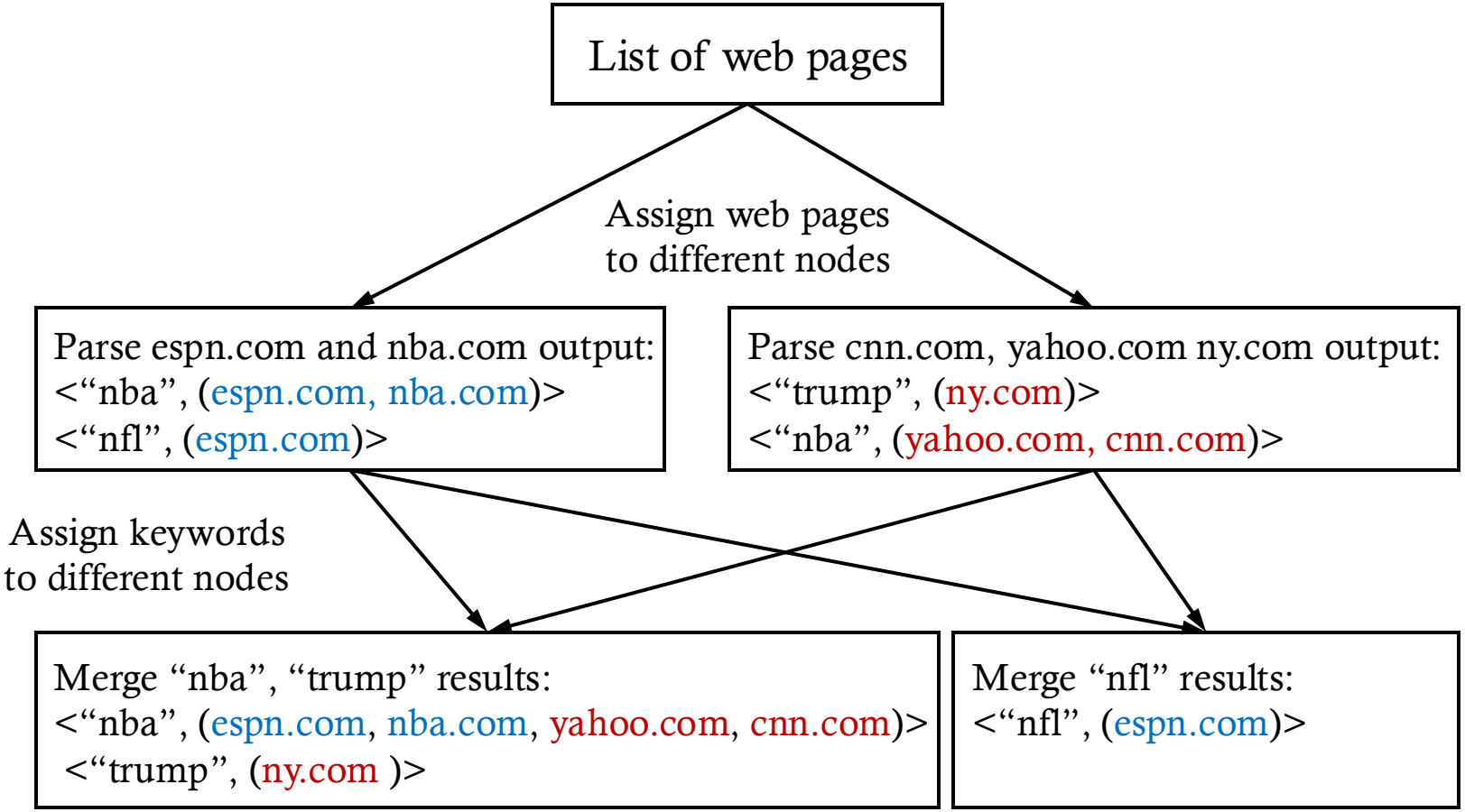
Parallel Web Page Indexing

- Need to parallelize indexing on multiple machines



Parallel Web Page Indexing

- What if we also want to parallelize the merge process?



```
// index a subset of web pages
index(List webpages) {
  Hash output = new Hash<string word,
                List<string url>>;

  foreach page p in webpages {
    for each word w in p {
      if (!output.exists (w))
        output{w} = new List<string>;
      // append web page for word w
      output{w}.push(URL(p));
    }
  }

  // partition data
  // send output to merge servers
  foreach word w in keys(output) {
   if (w in range [‘a’ – ‘d’])
    send(merge_serverA, output{w});
   else if (w in range [‘e’ – ‘h’]
    send(merge_serverB, output{w});
   .. ..
  }
}
```

```
merge() {
  // while any index server has data
  while (index_serverN sends data) {
    // receive data
    recv(index_serverN, output{w});
    // merge results in final_output
    final_output{w}.push(output{w});
  }
}
```

Problem

`final_output` stores results for all words, hat if it is so large that `merge()` runs out of memory?

```
// index a subset of web pages
index(List webpages) {
  Hash output = new Hash<string word,
                List<string url>>;

  foreach page p in webpages {
    for each word w in p {
      if (!output.exists (w))
        output{w} = new List<string>;
      // append web page for word w
      output{w}.push(URL(p));
    }
  }

  // partition data
  // send output to merge servers
  foreach word w in keys(output) {
   if (w in range [‘a’ – ‘d’])
    send(merge_serverA, output{w});
   else if (w in range [‘e’ – ‘h’]
    send(merge_serverB, output{w});
   .. ..
  }
}
```

```
merge() {
  // while any index server has data
  while (index_serverN sends data) {
    // receive and buffer data
    // in output, possibly on disk
    output += recv(index_serverN,
                   output{w});
  }
  // group output by word,
  // may require disk-based sort
  group_by_word(output);

  foreach w in keys(output) {

    // merge results in final_output
    final_output{w}.push(output{w});

    if (w != prev_w) {
      // done with prev_w
      // write prev_w output to disk
      write(final_output{prev_w});
    }
  }
}
```

Are we done?

Not So Fast!

- Need to handle failures
  - What if indexer is slow or fails?
    - Need to restart the indexer, mergers need to wait
  - What if merger fails?
    - Need to restart merger, need to wait for all mergers to finish
  - Need to ensure idempotent operation under all failures
    - Operation can be run multiple times, without additional side-effects
- What if partitioning is skewed?
  - E.g., frequency of words by initial letters is not the same
    - S (12%), C (9%), P, …. Y, Z (0.38%), X (0.09%)
  - Leads to load imbalance at merger
    - Need to repartition output of indexer for better performance

```
// index a subset of web pages
index(List webpages) {
  Hash output = new Hash<string word,
                         List<string url>>;

  foreach page p in webpages {
    for each word w in p {
      if (!output.exists (w))
        output{w} = new List<string>;
      // append web page for word w
      output{w}.push(URL(p));
    }
  }

  // partition data
  // send output to merge servers
  foreach word w in keys output {
    ...
  }
}
```

```
merge() {
  // while any index server has data
  while (index_serverN sends data) {
    // receive and buffer data
    // in output, possibly on disk
    output += recv(index_serverN,
                   output{w});
  }
  // group output by word,
  // may require disk-based sort
  group_by_word(output);

  foreach w in keys(output) {
    // merge results in final_output
    final_output{w}.push(output{w});
  }
}
```

What if programmers only had to write code inside the boxes?

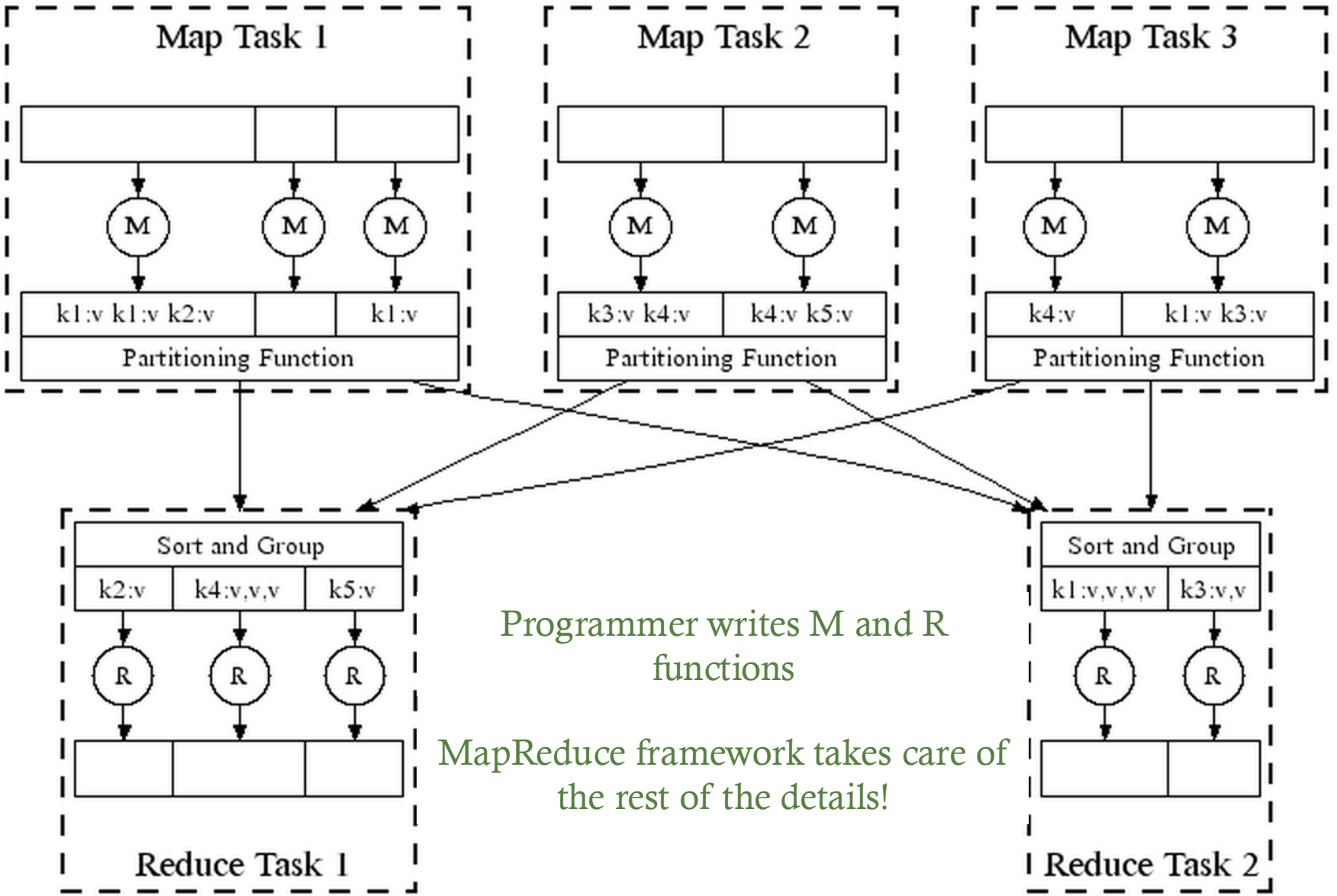
Solution: MapReduce

- Programming model for big data analytics
- Programmer writes two functions, called map and reduce

```
map(in_key, in_value)->list(out_key, intermediate_val)
```
Processes input key/value pair, produces set of intermediate pairs

```
reduce(out_key, list(intermediate_val))->list(out_key, outvalue)
```
Processes a set of intermediate key-values, produces value for each key

- Widely used model
  - At Google, used for indexing and many analytic jobs
  - Hadoop (open source version)
    - Used by > 50% of the Fortune 50 companies
    - Facebook analyzes half a PB per day using hadoop


Map Task 1
Map Task 2
Map Task 3
M
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Partitioning Function
Sort and Group
k2:v
k4:v,v,v
k5:v
k1:v,v,v,v
k3:v,v
R
Reduce Task 1
Reduce Task 2
Programmer writes M and R functions
MapReduce framework takes care of the rest of the details!

Web Page Indexing With MapReduce

```
// input: <url, web page content>
map(url, content) {
  for each word w in content {
    // output: <word, url>
    Emit(<w, url>);
  }
}

// input: <word, list of url>
reduce(char *word, List<url> l) {
  if (!final_output.exists(word))
    final_output{word} = new List<url>;

  // output: <word, list(url)>
  foreach url in l {
    final_output{word}.push(url);
  }
}
```

MapReduce Framework:

Mapper:

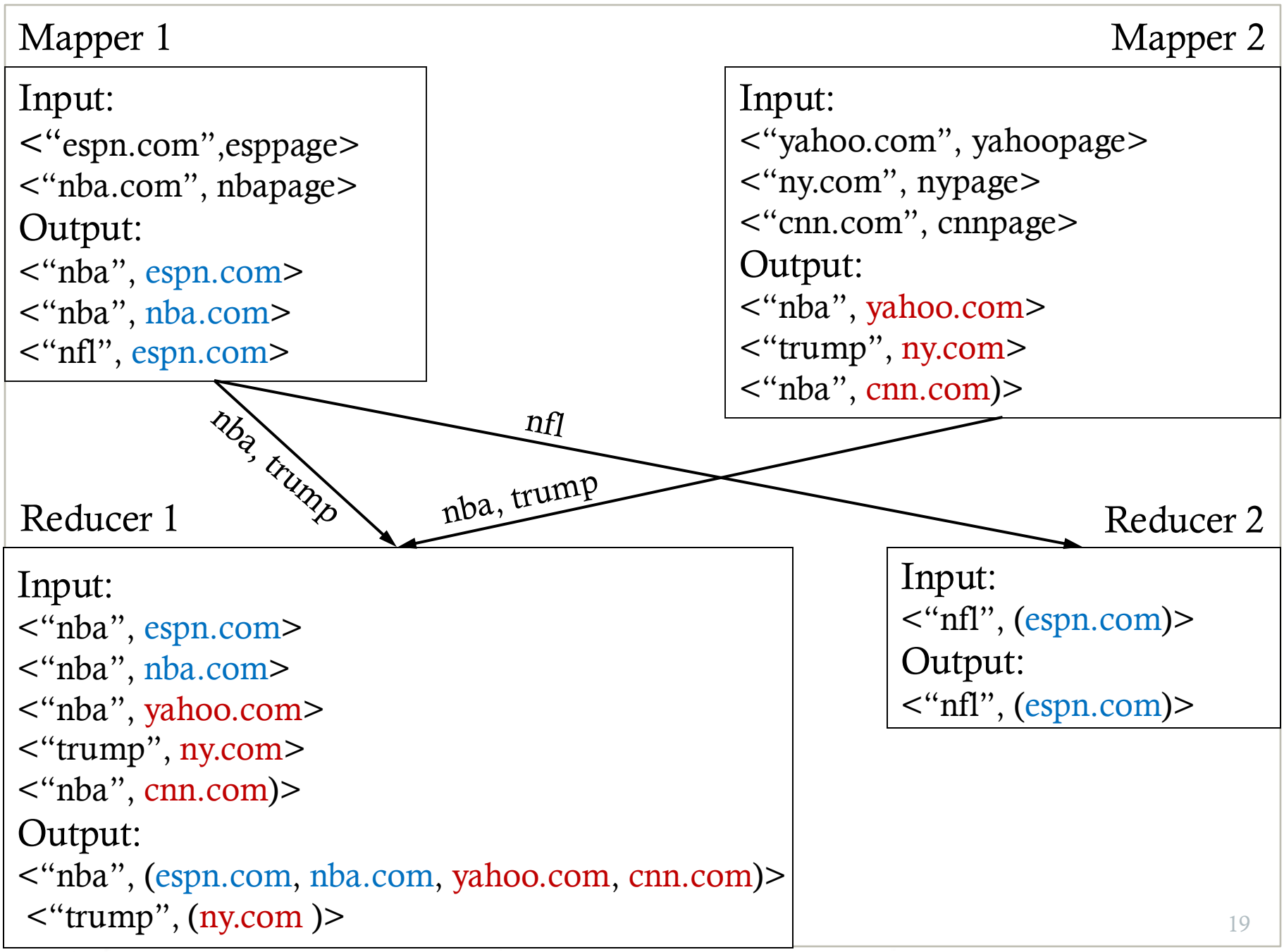
- Partitions intermediate output
- Sends same keys to same reducer

Reducer:

- Receives data
- Sorts and groups data by key

Master:

- Performs error handling


Mapper 1
Mapper 2
Input:
<“espn.com”,esppage>
<“nba.com”, nbapage>
Output:
<“nba”, espn.com>
<“nba”, nba.com>
<“nfl”, espn.com>
Input:
<“yahoo.com”, yahoopage>
<“ny.com”, nypage>
<“cnn.com”, cnnpage>
Output:
<“nba”, yahoo.com>
<“trump”, ny.com>
<“nba”, cnn.com)>
nba, trump
nfl
nba, trump
Reducer 1
Reducer 2
Input:
<“nba”, espn.com>
<“nba”, nba.com>
<“nba”, yahoo.com>
<“trump”, ny.com>
<“nba”, cnn.com)>
Output:
<“nba”, (espn.com, nba.com, yahoo.com, cnn.com)>
<“trump”, (ny.com)>
Input:
<“nfl”, (espn.com)>
Output:
<“nfl”, (espn.com)>

Reverse Web Links With MapReduce

```
// input: <url, web page content>
map(url, content) {
  for each target_url in content {
    // output: <target_url, url>
    Emit(<target_url, url>);
  }
}

// input: <target_url, list of url>
reduce(target_url, List<url>l) {
  if (!final_output.exists(target_url))
    final_output{target_url} = new List<url>;

  // output: <target_url, list(url)>
  foreach url in l {
    final_output{target_url}.push(url);
  }
}
```

Just need to replace word with target_url

Locality optimization:
map/reduce tries to run worker on machine storing the file split

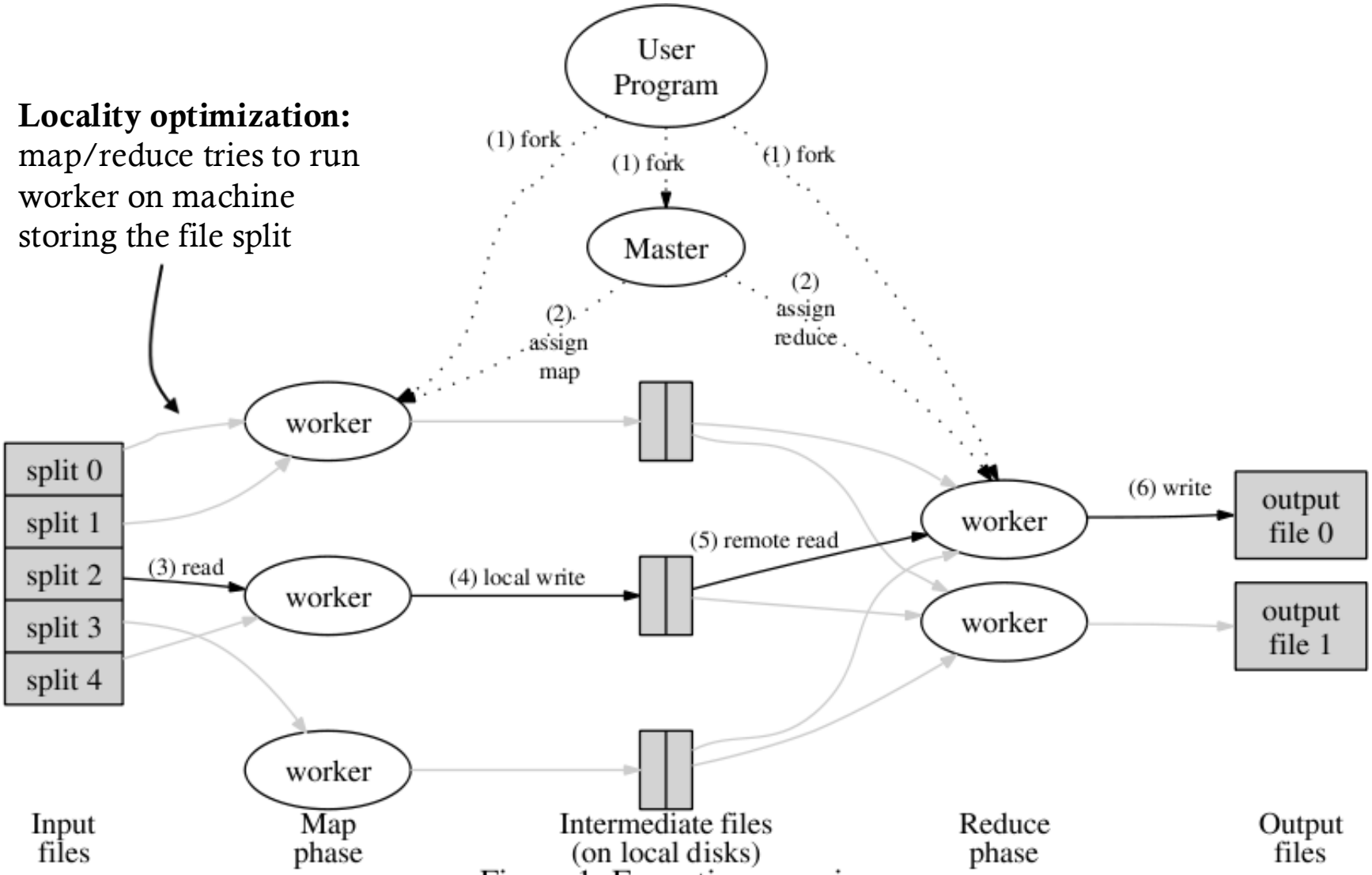


Figure 1: Execution overview

Handling Failures

- Machine failures are common in large distributed systems
  - "One node crashes per day in a 10K node cluster" - Jeff Dean
- Worker failure
  - Master detects worker failure via periodic heartbeats
  - Re-execute map/reduce tasks whose results are not available
- Master failure
  - Single point of failure
  - Master writes periodic checkpoints
  - Another master started from the last checkpointed state
- Google: Lost 1600 of 1800 machines once, but finished fine!

Refinement: Redundant Execution

- Slow workers significantly lengthen completion time
  - Called “Stragglers”
- Maybe caused by
  - Other jobs consuming resources on machine
  - Bad disks with soft errors transfer data very slowly
  - Software bugs
- Solution
  - Near end of phase, spawn backup copies of tasks
  - Whichever one finishes first “wins”
  - Doesn’t cause overhead if stragglers don’t exist

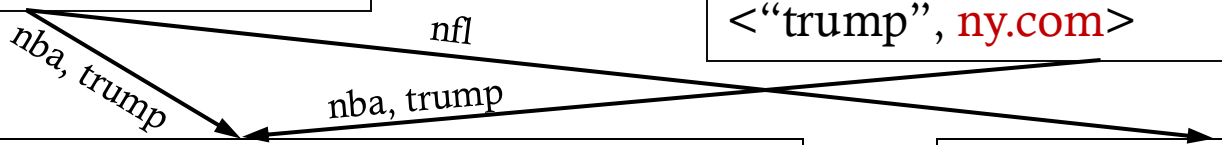
Refinement: Saving Network Bandwidth with Local Reduce

Mapper 1

Input:
<"espn.com",esppage>
<"nba.com", nbapage>
Output:
<"nba", (espn.com, nba.com)>
<"nfl", espn.com>

Mapper 2

Input:
<"yahoo.com", yahoopage>
<"ny.com", nypage>
<"cnn.com", cnnpage>
Output:
<"nba", (yahoo.com, cnn.com)>
<"trump", ny.com>



Reducer 1

Input:
<"nba", (espn.com, nba.com)>
<"trump", ny.com>
<"nba", (yahoo.com, cnn.com)>
Output:
<"nba", (espn.com, nba.com, yahoo.com, cnn.com)>
<"trump", (ny.com)>

Reducer 2

Input:
<"nfl", (espn.com)>
Output:
<"nfl", (espn.com)>

Various Advancements

- Master can become bottleneck
  - Split functionality of master
    - Scheduling, monitoring, recovery, etc.
  - Only scheduler is centralized
- I/O on intermediate results is slow
  - Buffer intermediate result in memory
- Other programming models
  - E.g., SQL on distributed systems (HIVE)
- More details: "MapReduce: Simplified Data Processing on Large Clusters". Jeff Dean and Sanjay Ghemawat, OSDI'04