

---

# COMP 322: Fundamentals of Parallel Programming

## Lecture 9: Map Reduce

Vivek Sarkar, Shams Imam  
Department of Computer Science, Rice University

Contact email: [vsarkar@rice.edu](mailto:vsarkar@rice.edu)

<http://comp322.rice.edu>

---

COMP 322

Lecture 9

1 February 2016



### Worksheet #8: Classifying different versions of parallel search algorithm

---

Enter "YES" or "NO", as appropriate, in each box below

Example: String Search variation	Data Race Free?	Functionally Deterministic?	Structurally Deterministic?
V1: Count of all occurrences	YES	YES	YES
V2: Existence of an occurrence	NO	YES	YES
V3: Index of any occurrence	NO	NO	YES
V4: Optimized existence of an occurrence: do not create more async tasks after occurrence is found	NO	YES	NO
V5: Optimized index of any occurrence: do not create more async tasks after occurrence is found	NO	NO	NO



# Streaming data requirements have skyrocketed

---

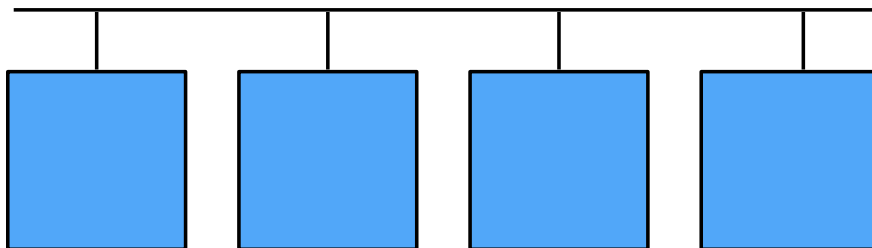
- AT&T processes roughly 30 petabytes per day through its telecommunications network
- Google processed roughly 24 petabytes per day in 2009
- Facebook, Amazon, Twitter, etc, have comparable throughputs
- Two Sigma maintains over 100 teraflops of private computing power, continuously computing over 11 petabytes of quantitative data
- In comparison, the IBM Watson knowledge base stored roughly 4 terabytes of data when winning at Jeopardy



## Parallelism enables processing of big data

---

- Continuously streaming data needs to be processed at least as fast as it is accumulated, or we will never catch up
- The bottleneck in processing very large data sets is dominated by the speed of disk access
- More processors accessing more disks enables faster processing



# Parallelism enables “Cloud Computing” as a Utility

---

- Offers computing, storage, communication at pennies per hour
  - *Leverage Parallelism to Achieve Energy-Efficient High Performance*
- No premium to scale:
  - 1000 computers @ 1 hour
  - = 1 computer @ 1000 hours
- Illusion of infinite scalability to cloud user
  - *As many computers as you can afford*
- Leading examples: Amazon Web Services (AWS), Google App Engine, Microsoft Azure
  - *Economies of scale pushed down datacenter costs by factors of 3-8X*
  - *Traditional data centers utilized 10% - 20%*
  - *Make profit offering pay-as-you-go use service at less than your costs for as many computers as you need*
  - *Strategic capability for company's needs*
- Challenge: portable and scalable parallelism at cloud scale
  - *One solution: leverage functional programming with MapReduce pattern*



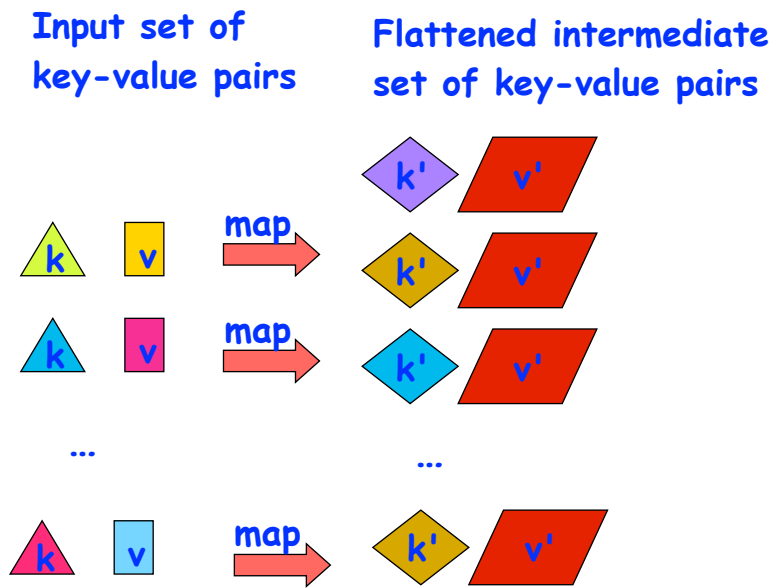
## MapReduce Pattern

---

- Apply **Map** function **f** to user supplied record of key-value pairs
- Compute set of intermediate key/value pairs
- Apply **Reduce** operation **g** to all values that share same key to combine derived data properly
  - Often produces smaller set of values*
- User supplies Map and Reduce operations in functional model so that the system can parallelize them, and also re-execute them for fault tolerance



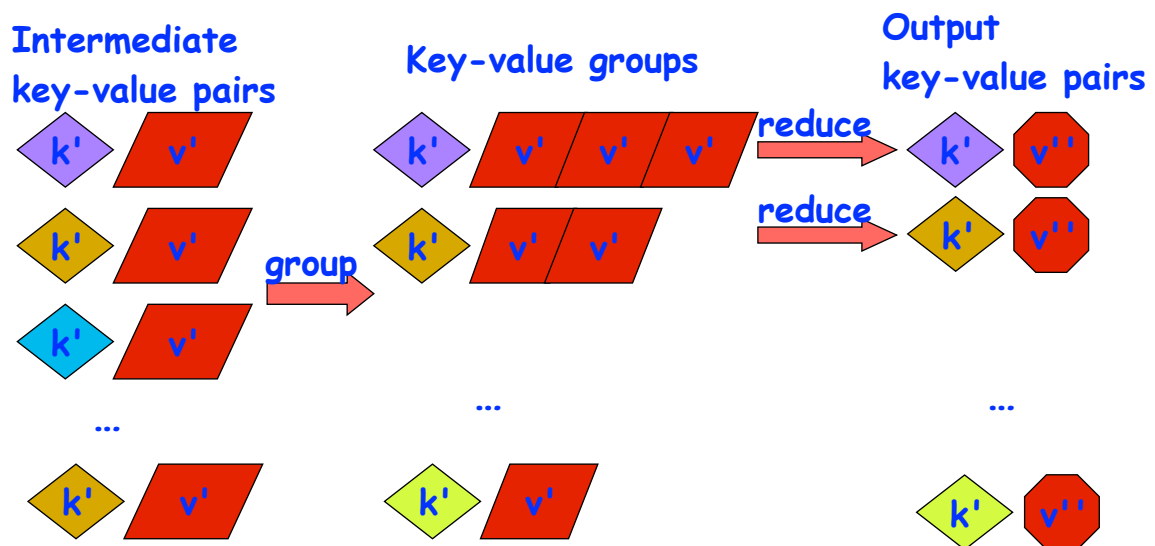
# MapReduce: The Map Step



Source: <http://infolab.stanford.edu/~ullman/mining/2009/mapreduce.ppt>



# MapReduce: The Reduce Step



Source: <http://infolab.stanford.edu/~ullman/mining/2009/mapreduce.ppt>



## Map Reduce: Summary

---

- Input set is of the form  $\{(k_1, v_1), \dots, (k_n, v_n)\}$ , where  $(k_i, v_i)$  consists of a key,  $k_i$ , and a value,  $v_i$ .
  - Assume that the key and value objects are immutable, and that equality comparison is well defined on all key objects.
- Map function  $f$  generates sets of intermediate key-value pairs,  $f(k_i, v_i) = \{(k_1', v_1'), \dots, (k_m', v_m')\}$ . The  $k_j'$  keys can be different from  $k_i$  key in the in of the map function.
  - Assume that a flatten operation is performed as a post-pass after the map operations, so as to avoid dealing with a set of sets.
- Reduce operation groups together intermediate key-value pairs,  $\{(k', v_j')\}$  with the same  $k'$ , and generates a reduced key-value pair,  $(k', v'')$ , for each such  $k'$ , using reduce function  $g$



## Google Uses MapReduce For ...

---

- **Web crawl**: Find outgoing links from HTML documents, aggregate by target document
- **Google Search**: Generating inverted index files using a compression scheme
- **Google Earth**: Stitching overlapping satellite images to remove seams and to select high-quality imagery
- **Google Maps**: Processing all road segments on Earth and render map tile images that display segments
- More than 10,000 MR programs at Google in 4 years, run 100,000 MR jobs per day (2008)



## MapReduce: State of Practice

---

- Apache Hadoop now dominates use of the MapReduce framework
- Often, Hadoop map and reduce functions are *no longer written directly*
  - *Instead, a user writes a query in a very high level language and uses another tool to compile the query into map/reduce functions!*
    - *Hive (another Apache project) compiles SQL queries into map/reduce*
    - *Pig (yet another Apache project) compiles direct relational algebra into map/reduce*



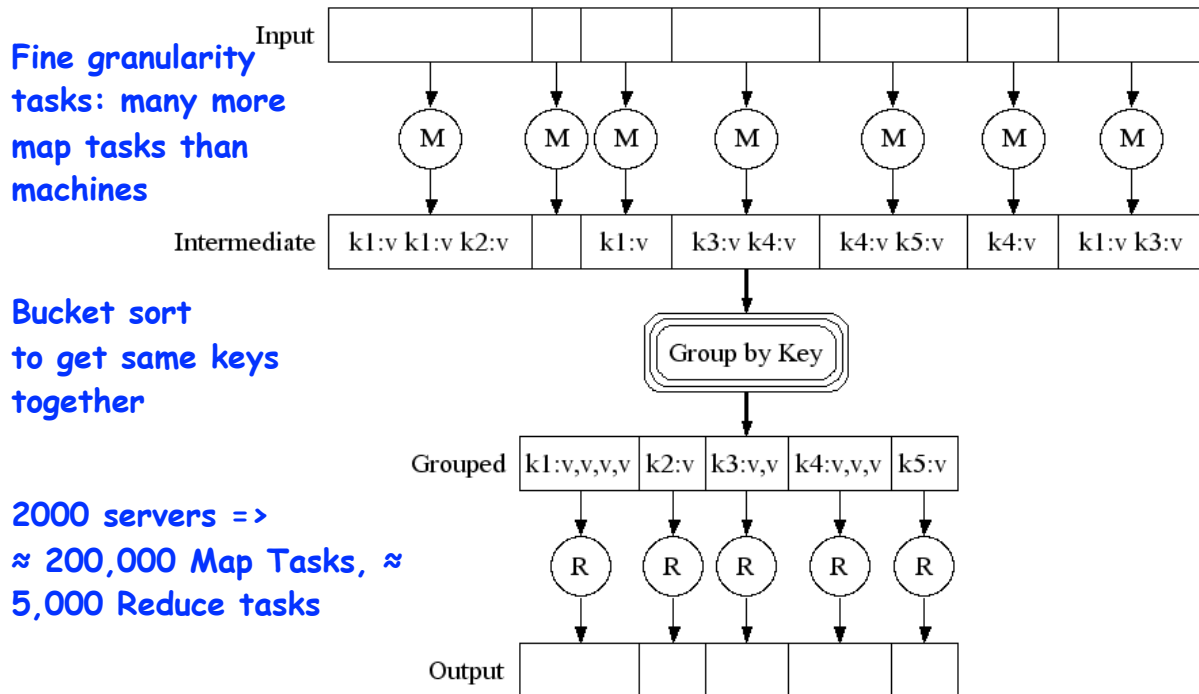
## MapReduce: State of Practice

---

- Eventually, users started realizing that a much larger class of algorithms could be expressed as an iterative sequence of map/reduce operations
  - *Many machine learning algorithms fall into this category*
- Tools started to emerge to enable easy expression of multiple map/reduce operations, along with smart scheduling
- Apache Spark: General purpose functional programming over a cluster
  - *Caches results of map/reduce operations in memory so they can be used on subsequent iterations without accessing disk each time*
  - *Tends to be 10-100 times faster than Hadoop for many applications*



# MapReduce Execution



# WordCount example

In: set of words

Out: set of (word,count) pairs

Algorithm:

1. For each in word  $W$ , emit  $(W, 1)$  as a key-value pair (map step).
  2. Group together all key-value pairs with the same key (reduce step).
  3. Perform a sum reduction on all values with the same key(reduce step).
- All map operations in step 1 can execute in parallel with only local data accesses
  - Step 2 may involve a major reshuffle of data as all key-value pairs with the same key are grouped together.
  - Step 3 performs a standard reduction algorithm for all values with the same key, and in parallel for different keys.



# PseudoCode for WordCount

```
1. <String, Integer> map(String inKey, String inValue):
2.   // inKey: document name
3.   // inValue: document contents
4.   for each word w in inValue:
5.     emitIntermediate(w, 1) // Produce count of words
6.
7. <Integer> reduce(String outKey, Iterator<Integer> values):
8.   // outKey: a word
9.   // values: a list of counts
10.  Integer result = 0
11.  for each v in values:
12.    result += v // the value from map was an integer
13.  emit(result)
```

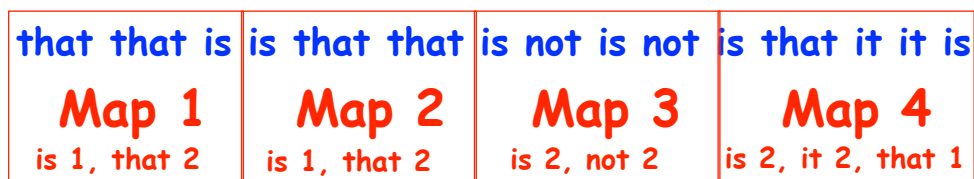
15

COMP 322, Spring 2016 (V. Sarkar, S. Imam)



## Example Execution of WordCount Program

### Distribute



### Shuffle



### Collect

is 6; it 2; not 2; that 5

16

COMP 322, Spring 2016 (V. Sarkar, S. Imam)

