Dealing with Failures

- Map worker failure
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker
- Reduce worker failure
  - Only in-progress tasks are reset to idle
  - Reduce task is restarted
- Master failure
  - MapReduce task is aborted and client is notified

How many Map and Reduce jobs?

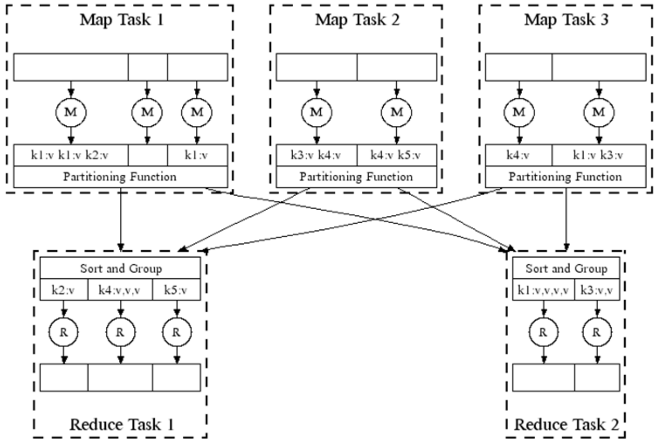
- M map tasks, R reduce tasks
- Rule of a thumb:
  - Make M much larger than the number of nodes in the cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds up recovery from worker failures
  - Usually R is smaller than M, because output is spread across R files

Refinements: Backup Tasks

- Problem
  - Slow workers significantly lengthen the job completion time:
    - Other jobs on the machine
    - Bad disks
    - Weird things
- Solution
  - Near end of phase, spawn backup copies of tasks
    - Whichever one finishes first “wins”
- Effect
  - Dramatically shortens job completion time

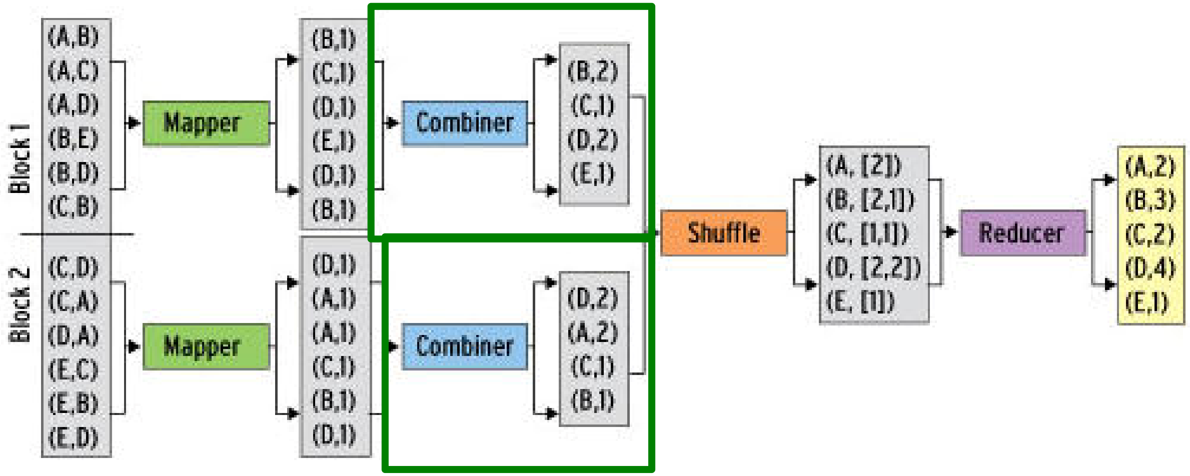
Refinement: Combiners

- Often a Map task will produce many pairs of the form (k,v1), (k,v2), … for the same key k
  - E.g., popular words in the word count example
- Can save network time by pre-aggregating values in the mapper:
  - combine(k, list(v1)) → v2
  - Combiner is usually same as the reduce function
- Works only if reduce function is commutative and associative 交换律和结合律, e.g., sum



Refinement: Combiners

- Back to our word counting example:
  - Combiner combines the values of all keys of a single mapper (single machine):



  - Much less data needs to be copied and shuffled!

Refinement: Partition Function

- Want to control how keys get partitioned
  - Inputs to map tasks are created by contiguous splits of input file
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function:
  - hash(key) mod R
- Sometimes useful to override the hash function:
  - E.g., hash(hostname(URL)) mod R ensures URLs from a host end up in the same output file

Example: Host size

- Suppose we have a large web corpus (语料库) with a metadata file formatted as follows:
  - Each record of the form: (URL, size, date, …)
- We want to: For each host(not each URL), find the total number of bytes
  - That is, the sum of the page sizes for all URLs from that particular host
- Map: For each record, output(hostname(URL),size)
- Reduce: sum the size of each host

Example: Language Model

- Statistical machine translation:
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents
- Very easy with MapReduce:
  - Map:
    - Extract (5-word sequence, count) from document
  - Reduce:
    - Combine the counts

Example: Join By Map-Reduce

- Compute the natural join R(A,B) ⋈ S(B,C). R and S are each stored in files. Tuples are pairs (a,b) or (b,c)
- **Map**: (b,(R,a))for each tuple on R; (b,(S,c))for each tuple on S
- **Reduce**:same key with (R,a)) or (S,c), then output only (a,c). key is irrelevant.

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

⋈

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

Pointers and Further Reading

Reading

- Jeffrey Dean and Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters
  - http://labs.google.com/papers/mapreduce.html

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System
  - http://labs.google.com/papers/gfs.html

Resources

- Hadoop Wiki
  - Introduction
    - http://wiki.apache.org/lucene-hadoop/
  - Getting Started
    - http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop
  - Map/Reduce Overview
    - http://wiki.apache.org/lucene-hadoop/HadoopMapReduce
    - http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses
  - Eclipse Environment
    - http://wiki.apache.org/lucene-hadoop/EclipseEnvironment
- Javadoc
  - http://lucene.apache.org/hadoop/docs/api/

Resources

- Releases from Apache download mirrors
  - http://www.apache.org/dyn/closer.cgi/lucene/hadoop/
- Nightly builds of source
  - http://people.apache.org/dist/lucene/hadoop/nightly/
- Source code from subversion
  - http://lucene.apache.org/hadoop/version_control.html

Further Reading

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]