Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value

pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key.

Mapreduce 有两个部分 第一部分就是映射 第二部分是合并所有的映射（之前的key – value）

Mapreduce大致用两种mapreduce实现

The major contributions of this work are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

使用的Programming model

*Map*, written by the user, takes an input pair and produces a set of *intermediate* key/value pairs. The MapReduce

library groups together all intermediate values associatedwith the same intermediate key I and passes them to the *Reduce* function and merge these sets of values.

几个使用mapreduce的应用场景

1) Machines are typically dual-processor x86 processors running Linux, with 2-4 GB of memory per machine. 分布式grep, grep就是linux中的一个命令 用于查找文件里符合条件的字符串

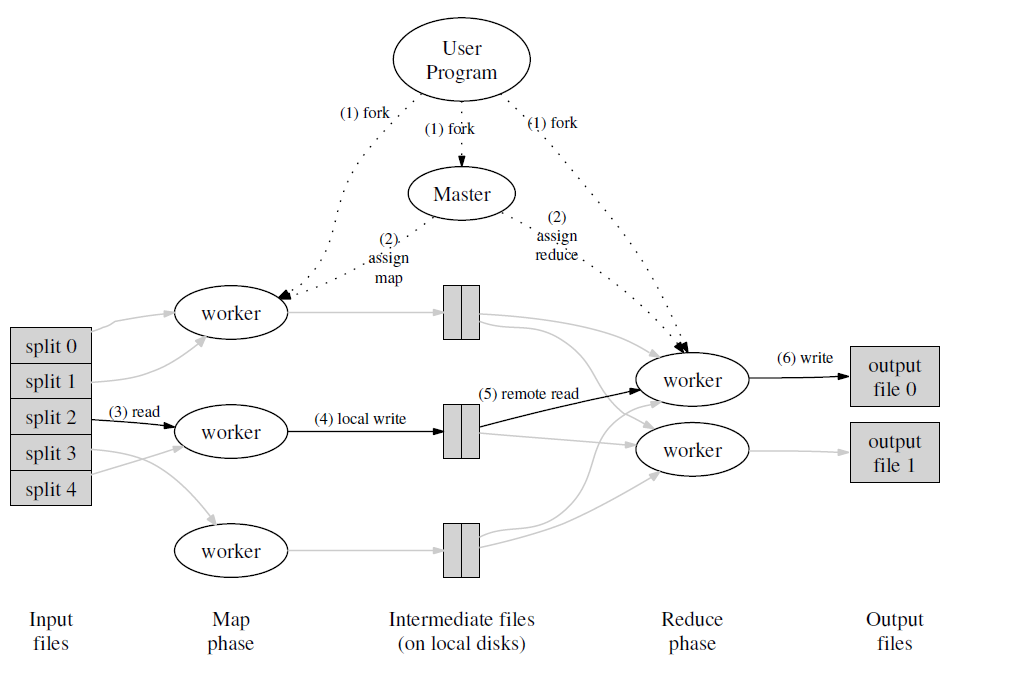
(2) Commodity networking hardware is used – typically either 100 megabits/second or 1gigabit/second at the machine level, but averaging considerably less in overall bisection bandwidth.

(3) A cluster consists of hundreds or thousands of machines, and therefore machine failures are common.

(4) Storage is provided by inexpensive IDE disks attached directly to individual machines. A distributed file system [8] developed in-house is used to manage the data stored on these disks. The file system uses replication to provide availability and reliability on top of unreliable

hardware.

(5) Users submit jobs to a scheduling system. Each job consists of a set of tasks, and is mapped by the scheduler to a set of available machines within a cluster.



Execution overview

* User program tells master it wants to run a map reduce job
* Master assign workers based on where the files are stored
* Apply map functions to the file chunks (store results on local disk)
* Call the user reduce function per key with the list of values for that key to aggregate the results

Detail Fault Tolerance

Worker failure

Master detect failure periodically

Re-executre Map tasks

Re-execute in progress Reduce tasks

Master failure

Single master->unlikely

Abort

Detail ---Locality

Network bandwidths is a scare resource

Run on GFS(64 MB blocks, several replica)

Map tasks scheduled so GFS input block replica are on same machine or same rack

Detail --- Combine function

Network bandwidth is a scare resource

Word Counting example

Hundreds or thousands of records of the form<the, 1>

Merging the data before sent over the network <the, 100>

Detail--- Task Grandularity

How many Maps? How many Reduces?

The more, the better

Minimizes time for fault recovery

Can pipeline shuffling with map execution

Dynamic load balancing

In practice

Choose Map : task 16 MB – 64 MB (GFS block size)

Choose Reduce : a small multiple of the number of worker

200,000 map /5000 reduce tasks w/ 2000 machines

Detail---Backup tasks

“Staggler”--- slow workers

Bad disk

Other jobs consulting resources

Weird things : cache disabled ?

Solution : Near end of phase, backup tasks

Whichever one finishes first “wins”

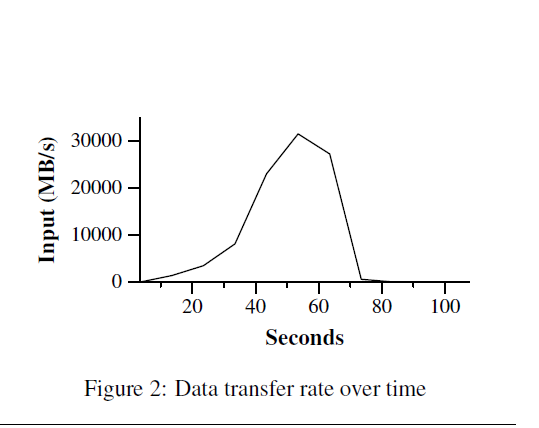
Detail- Skipping Bad Records

Records cause deterministic crashes

Best solutions to debug & fix, but not always possible

Solution : Detect and skip

If master sees two failures for same record



MR\_Grep

Locality optimization helps:

1800 machines read 1 TB of data at peak of ~31 GB / s

Without this, rack switches would limit to 10 GB / s

Startup overhead is significant for short jobs

Propagation of the program to all worker machine

MR\_Sort

Backup tasks

Failures

