BITS Pilani
Pilani Campus

Lecture-6
Big Data Systems
(SEZG522/CCZG522)

Slides: Courtesy:.Prof. Anindya

innovate achieve lead

BITS Pilani
Pilani Campus

Second Semester

2024-25

Lecture -6 Contents

- Top down design
- Types of parallelism
- MapReduce programming model
- See how a map reduce program works using Hadoop
- Iterative MapReduce

Top down design - sequential context

In the context of a sequential program

- Divide and conquer
  - It is easier to divide a problem into sub-problems and execute one by one
- A sub-problem definition may be left to the programmer in a sequential programming context

main()

f1() → f2() → f5()

f3()

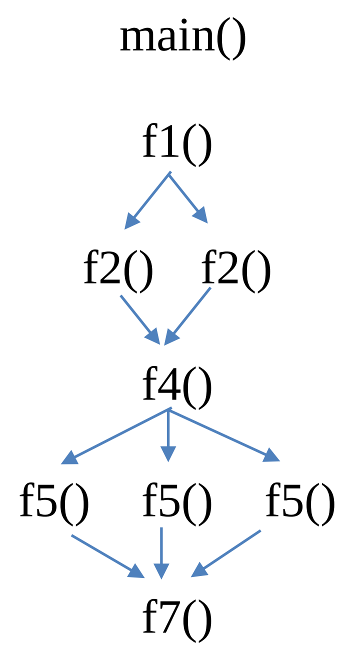
f4()

Top down design - parallel context

We cannot decompose the problem into sub-problems in anyway the programmer chooses to

Need to think about

- Each sub-problem needs to be assigned to a processor
  - Goal is to get the program work faster
- Divide the problem only when we can combine at the end into the final answer
  - Need to decide where to do the combination
  - Is there any parallelism in combination or is it sequential or trivial



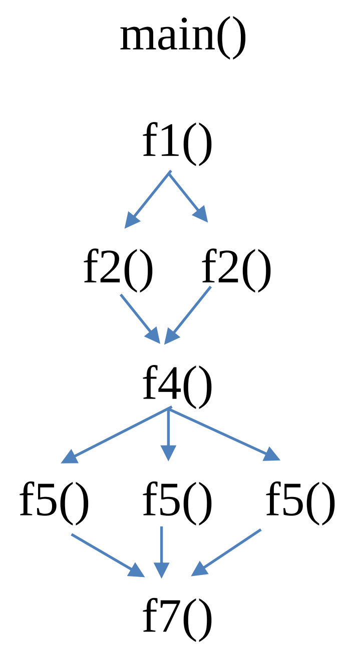
Deciding on number of sub-problems

In conventional top down design for sequential systems

- Keep number of sub-problems manageable
- Because need to keep track of them as computation progresses

In parallel system it is dictated by number of processors

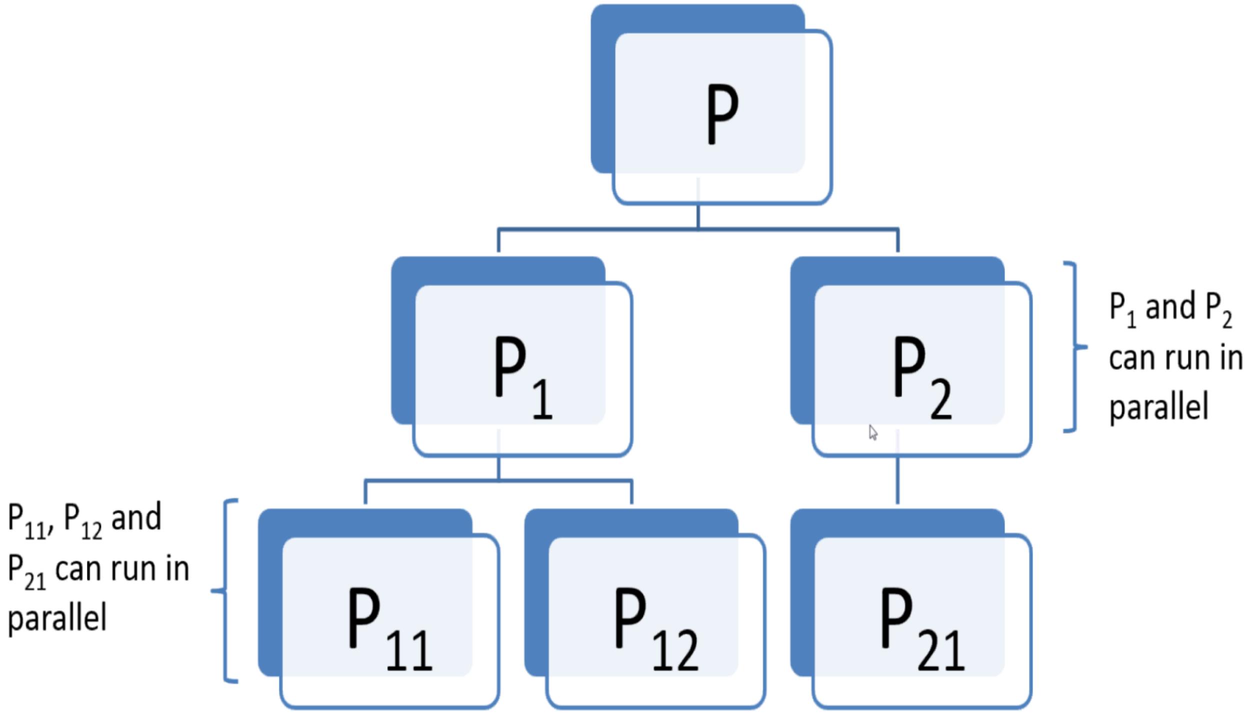
- Processor utilisation is the key
- If there are N processors, we can potentially have N sub-problems



How many processors ?

Top-down design

At each level problems need to run in parallel



Example 1 - Keyword search in list

Problem:

– Search for a key k in a sorted list Ls of size N

Data:

– Ls is stored in a distributed system with p processors each storing N/p items

Solution:

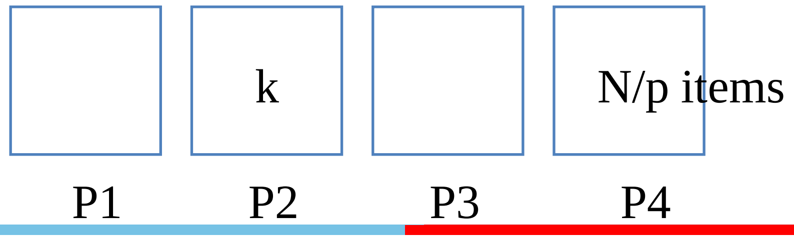
– Run binary search in each of the p processors in parallel

– Whichever processor finds k return (i, j) where ith processor has found key in jth position

– Combination: One or more positions are collected at processor 0

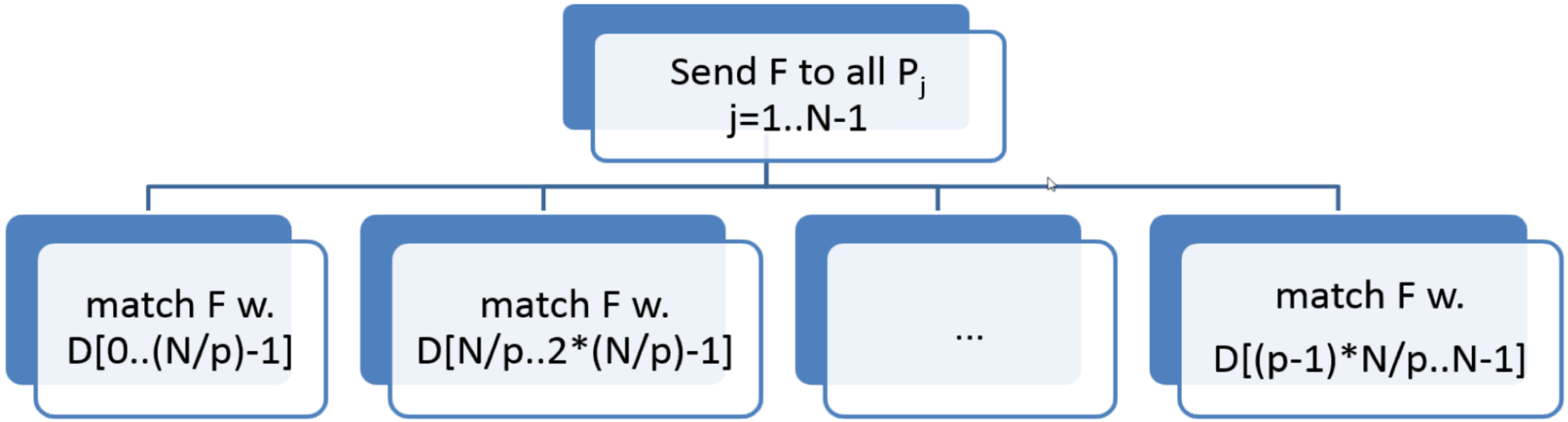
Speedup: p

Time complexity: O(Log(N/p))



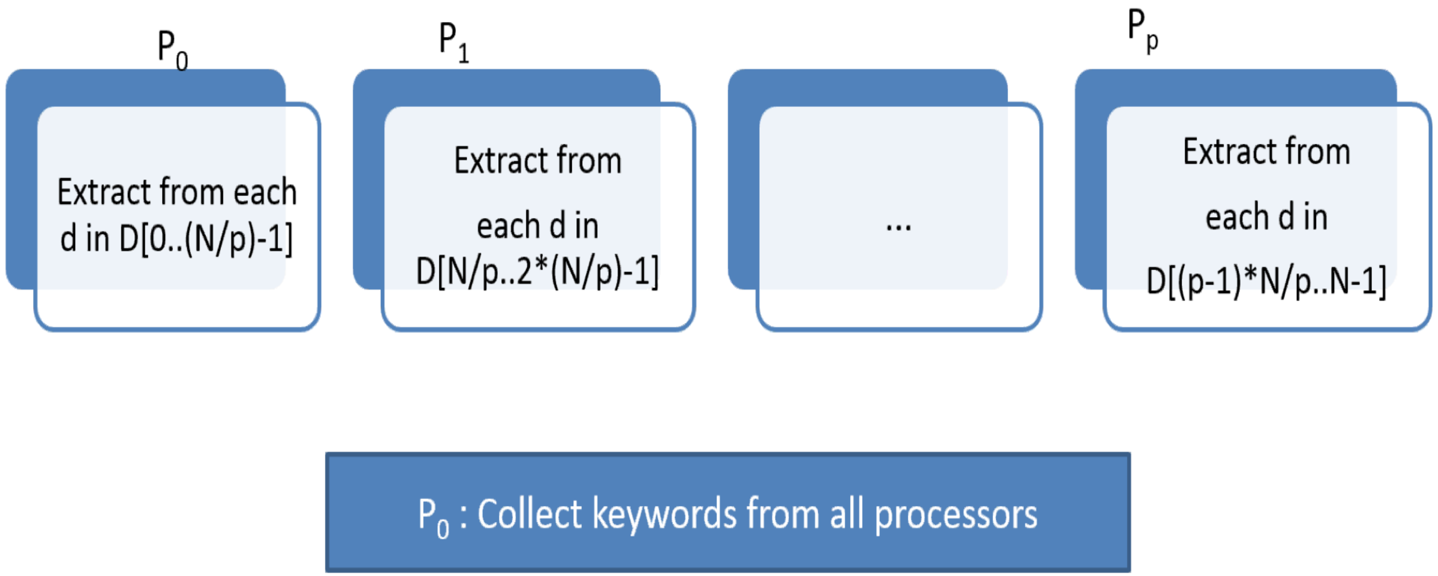
Example 2 - Fingerprint matching

- Find matches for a fingerprint F in a database of D prints
- Set of D prints is partitioned and evenly stored in a distributed database
- Partitioning is an infrequent activity - only when many new entries in database
- Search is the frequent activity
- Speed up p
- Time complexity O(N/p) given sequential search in every partition



Example 3: Document search

Find keywords from each document d in a distributed document collection D



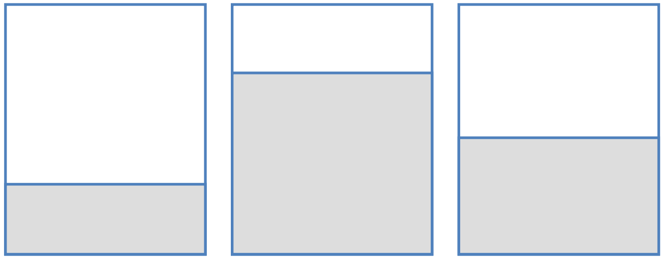
Data parallel execution model

Data is partitioned to multiple nodes / processors

- Try to make partitions equal or balanced

All processors execute the same code in parallel

- For homogenous nodes and equal amount of work, the utilization will be close to 100%
- Execution time is minimal
- Unbalanced data size / work or heterogenous nodes will lead to higher execution time



Where data parallelism is not possible

There are problems where you cannot divide the work

1. equally
2. independently to proceed in parallel

QuickSort(Ls, N)

– All N items in Ls have to be in memory of processor 0

Example : QuickSort

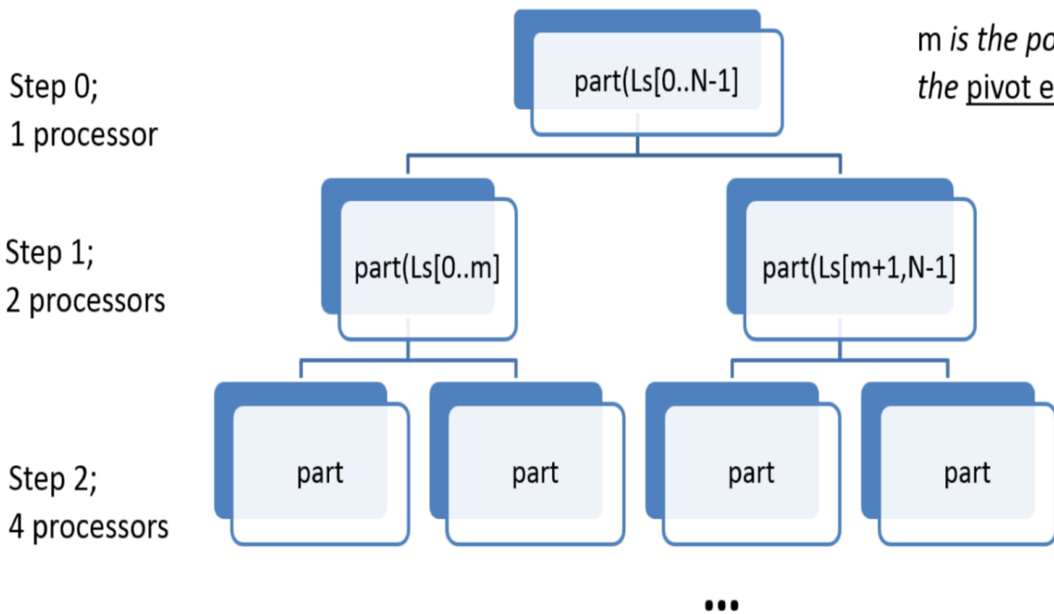
Pick a pivot element at position m to partition Ls and partition into sub-problems

Do that in each level

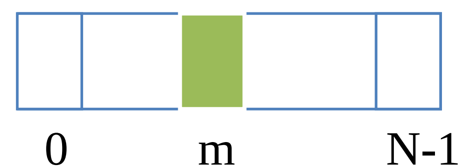
There is dependency on parent level to partition - not a single level problem (Log N or log p levels which ever is lower)

Choice of m cannot guarantee equal partition

- At a level one set of processors can get large partitions and another set small partitions
- Could be techniques to maintain balanced partitions and improve processor utilization uniformly



m is the position of the pivot element



you can't guarantee equal partitions based on D[m]

Tree parallel execution model for Quicksort parallel logic

In step j

- For each processor p from 1 to 2^{j-1} do
  - partition Ls_p into Ls_{p1} and Ls_{p2}
  - assign Ls_{p1} and Ls_{p2} to processors $2*p-1$ and $2*p$
  - $j = j + 1$
- repeat until $2^j == N$

- Depends on how good is the partition - at random ?
  - May be over long term for large lists and many processors
- Time taken may be as bad as sequential with bad partitioning

Tree parallelism summary

Dynamic version of divide and conquer - partitions are done dynamically

Division of problem into sub-problems happens execution time

- Sub-problem is identical in structure to the larger problem
- What is the division step ?
  - In quick sort it was picking m to split into 2 sub-problems
  - Division / partitioning logic is important to find almost equal sub-problems

If problem is divided into k sub-problems

- then in $Log_k N$ steps needed if N processors execute in parallel
- If p = N then work gets done in Log(N) time with each list item assign to one processor finally

What if we assign p processors with

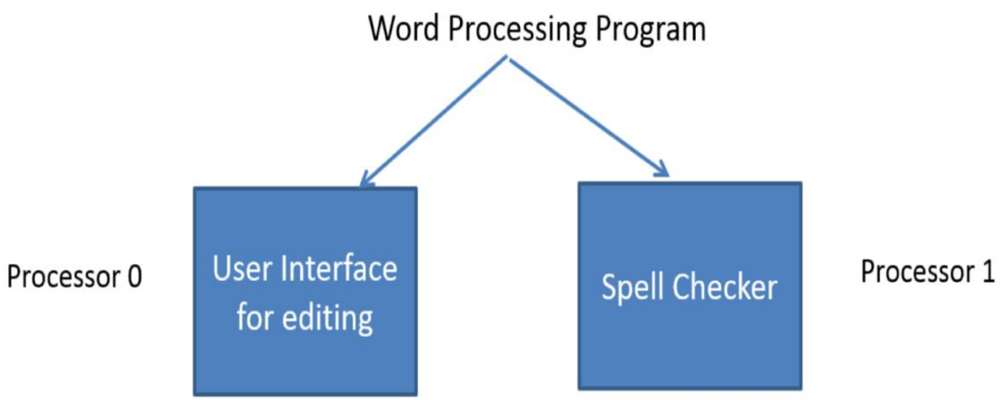
- p < N : so that all processors are utilised
- p > N : under-utilised processors

Task parallelism - Example 1 : Word processor

Parallel tasks that work on the same data

- Unlike data and tree parallel Data doesn't need to be divided, the Task gets divided into sub-tasks
- May work on same data instance, else need to make data copies and keep them in sync

If on multiple core, different threads can execute tasks in parallel accessing same data instance in memory



Task parallelism - Example 2 : Independent statistics

Given a list Ls of numeric values find its mean, median and mode

Solution

- Independent tasks on same data
- Each task can find a statistic on Ls
- Run tasks in parallel

Task parallelism summary

- Identify sub-tasks based on functionality with no common function
  - In Tree and Data parallel the tasks are identical function
- Sub-tasks are not identified based on data
- Independent sub-tasks are executed in parallel
- Sub-tasks are often limited and known statically in advance
  - We know in a word processor what are the sub-tasks
  - We know in statistical analysis what functions we will run in advance
  - So limited parallelism scope - not scalable with more resources
  - In data or tree parallelism we can potentially get more parallelism with more data - more scalable with more resources at same time interval

Request parallelism

Problem

- Scalable execution of independent tasks in parallel
- Execute same code but in many parallel instances

Solution

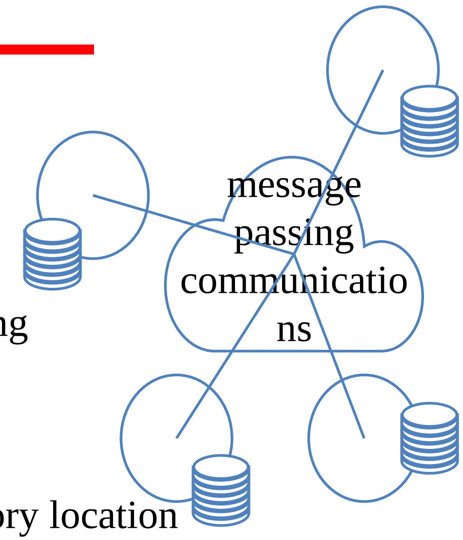
- On arrival, each request is serviced in parallel along with other existing tasks servicing prior requests
- Could be processing same or fixed data
- Request-reply pairs are independent of each other serviced by a different thread or process in the backend
  - There could be some application specific backend dependency, e.g. GET and POST on same data item

Systems Fit

- Servers in client-server models
- e.g. email server, HTTP web-server, cloud services with API interface, file / storage servers

Scalability metrics : Requests / time (throughput)

What happens in a loosely coupled distributed system



Divide

- No shared memory
- Memory / Storage is on separate nodes
- So any exchange of data or coordination between tasks is via message passing
- Divide the problem in a way that computation task can run on local data

Conquer / Merge

- In shared memory merge it is simpler with each process writing into a memory location
- In distributed
  - Need to collect data from the different nodes
  - In search example, it is a simpler merge to just collect result - so low cost
  - In quick sort, it is simple append whether writing in place for shared memory or sending a message
- Sometimes merges may become sequential
  - e.g. k-means - in each iteration (a) guess clusters in parallel to improve the clusters but (2) checking if we have found right clusters is sequential

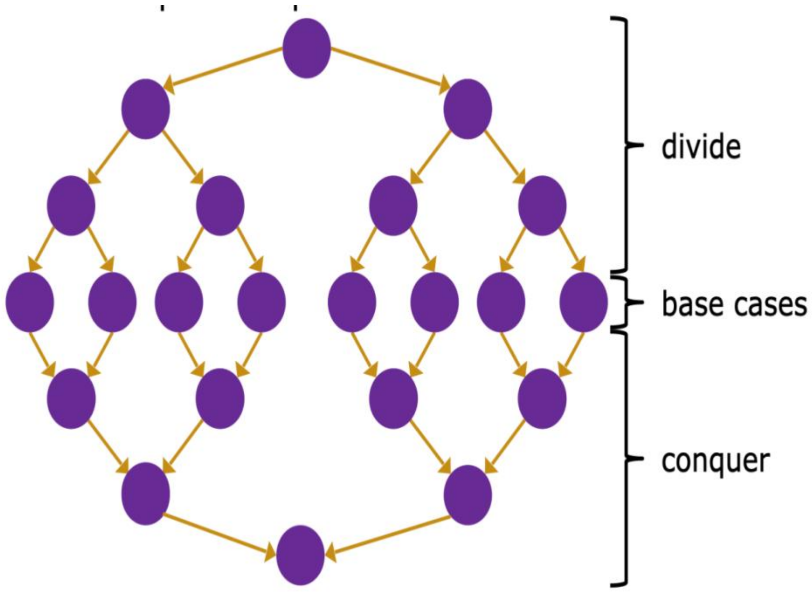
MapReduce in terms of Data and Tree parallelism

Map

- Data parallelism
- Divide a problem into sub-problems based on data

Reduce

- Inverse tree parallelism
- With every merge / reduce the parallelism reduces until we get one result
- Depending on the problem "reduce" step may be simple or sequential



Example: Word Count using MapReduce

map(key, value):

// key: document name; value: text of document

    for each word w in value:

        emit(w, 1)

reduce(key, values):

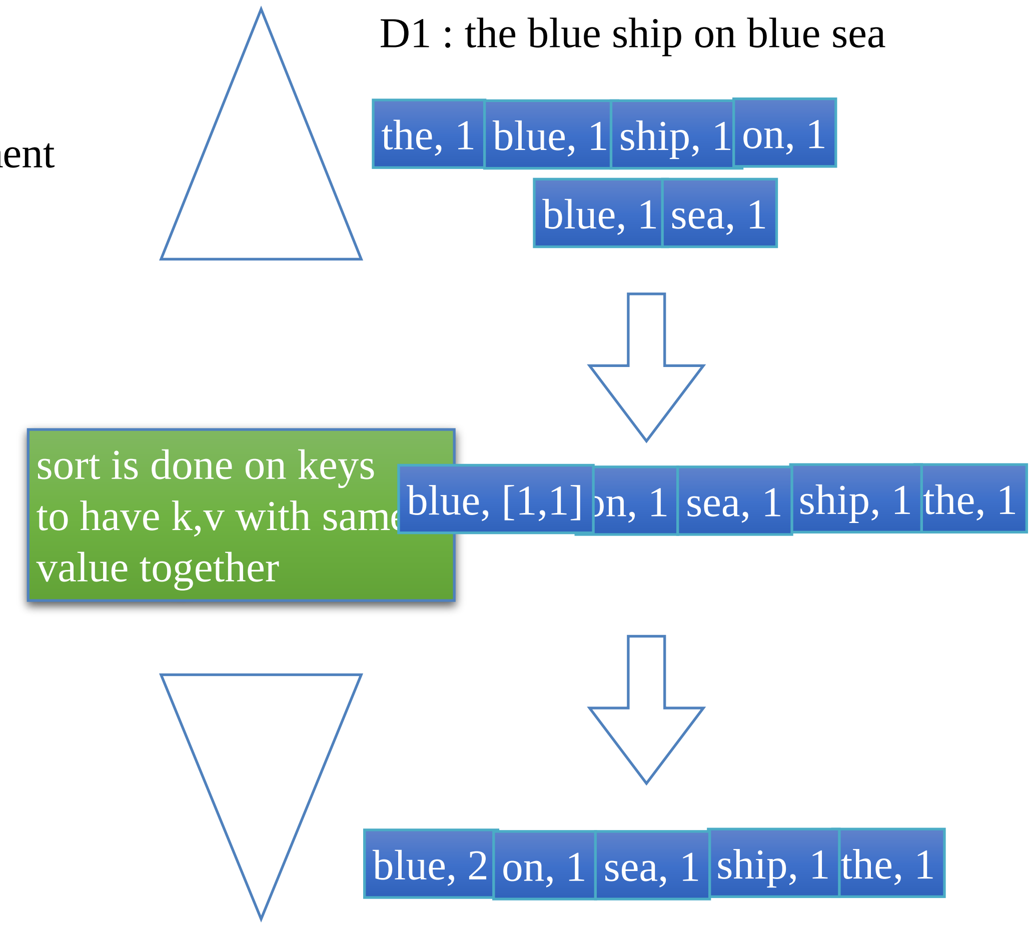
// key: a word; value: an iterator over counts

    result = 0

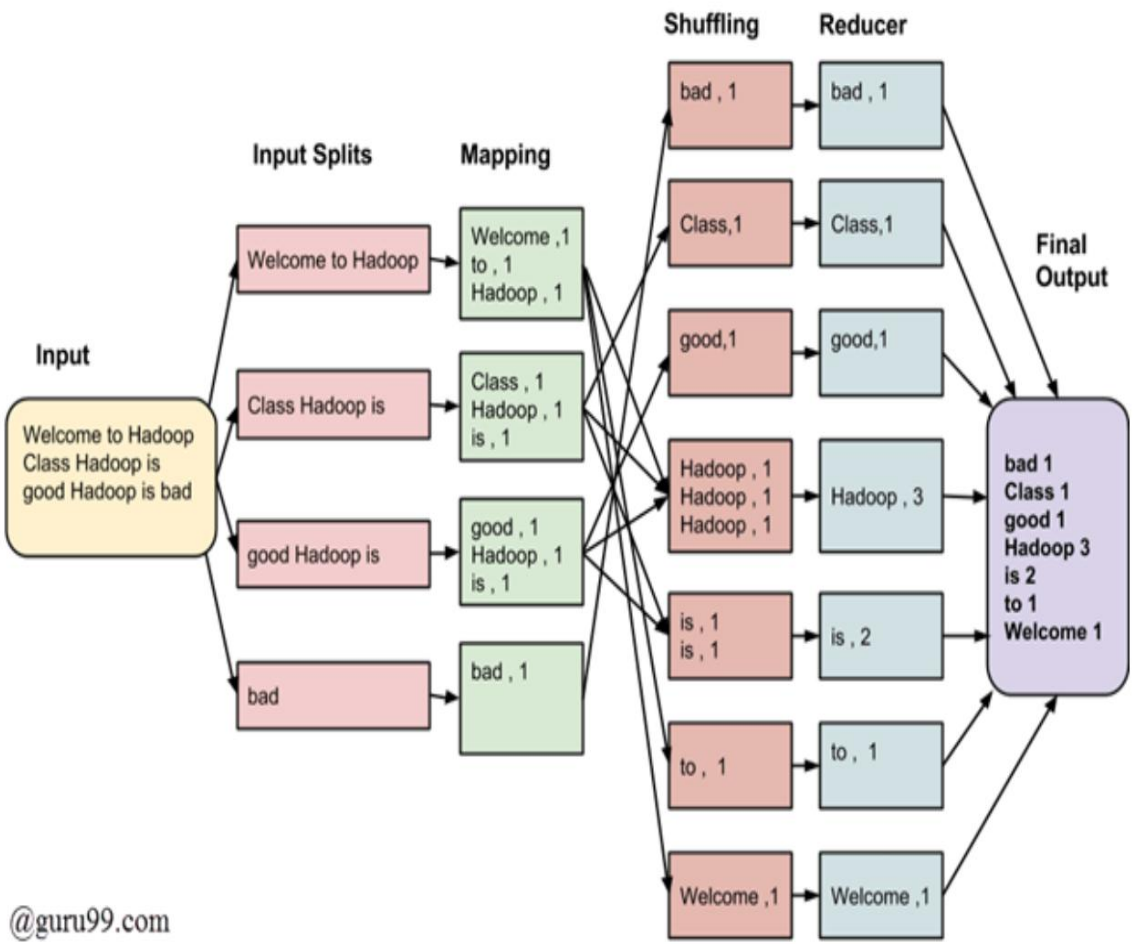
    for each count v in values:

        result += v

    emit(result)



Word count (2)

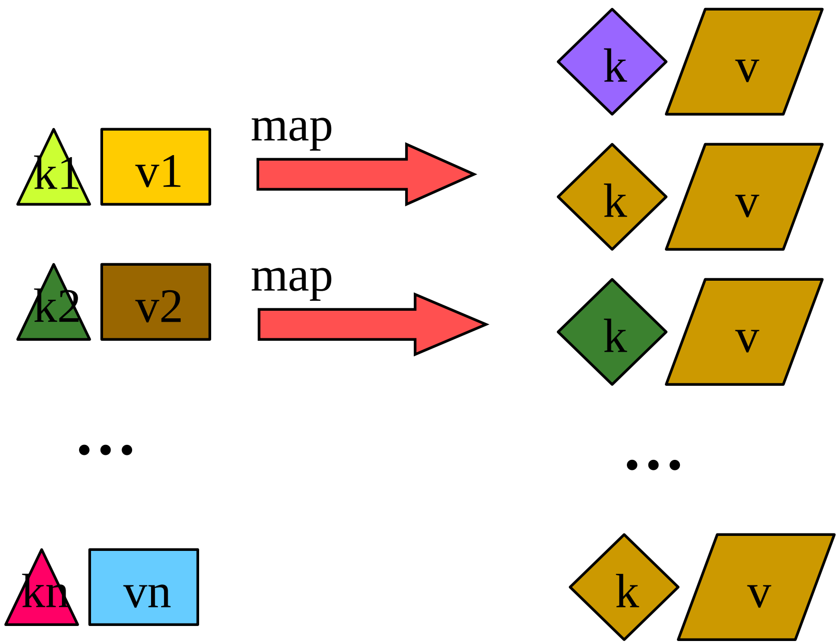


Input
Welcome to Hadoop
Class Hadoop is
good Hadoop is bad
Input Splits
Welcome to Hadoop
Class Hadoop is
good Hadoop is
bad
Mapping
Welcome ,1
to , 1
Hadoop , 1
Class , 1
Hadoop , 1
is , 1
good , 1
Hadoop , 1
is , 1
bad , 1
Shuffling
bad , 1
Class,1
good,1
Hadoop , 1
Hadoop , 1
Hadoop , 1
is , 1
is , 1
to , 1
Welcome ,1
Reducer
bad , 1
Class,1
good,1
Hadoop , 3
is , 2
to , 1
Welcome ,1
Final Output
bad 1
Class 1
good 1
Hadoop 3
is 2
to 1
Welcome 1
@guru99.com

MapReduce: The Map Step

Input
key-value pairs

Intermediate
key-value pairs

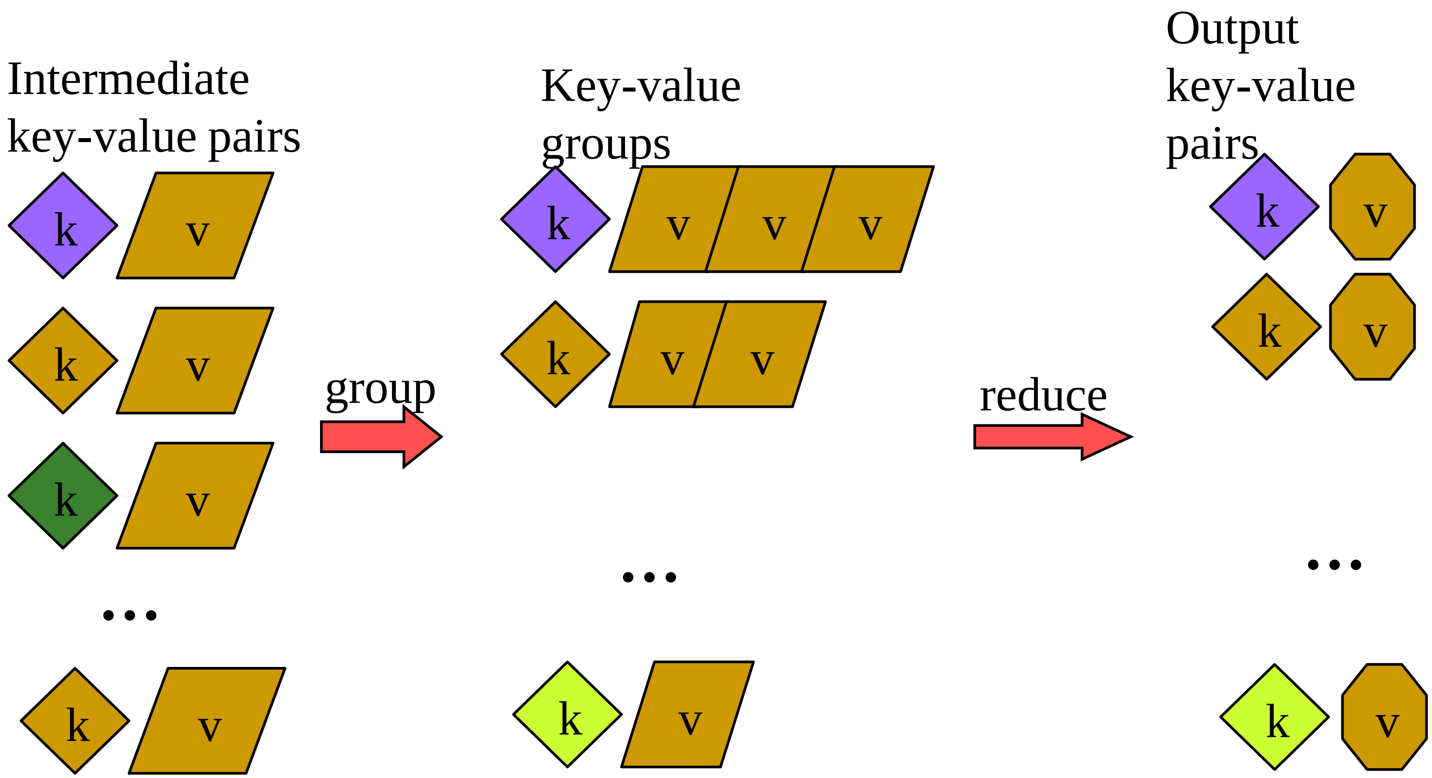


E.g. (doc—id, doc-content)

E.g. (word, wordcount-in-a-doc)

Adapted from Jeff Ullman's course slides

MapReduce: The Reduce Step



E.g.
(word, wordcount-in-a-doc)

(word, list-of-wordcount)
~ SQL Group by

(word, final-count)
~ SQL aggregation

Adapted from Jeff Ullman's course slides

Formal definition of a MapReduce program

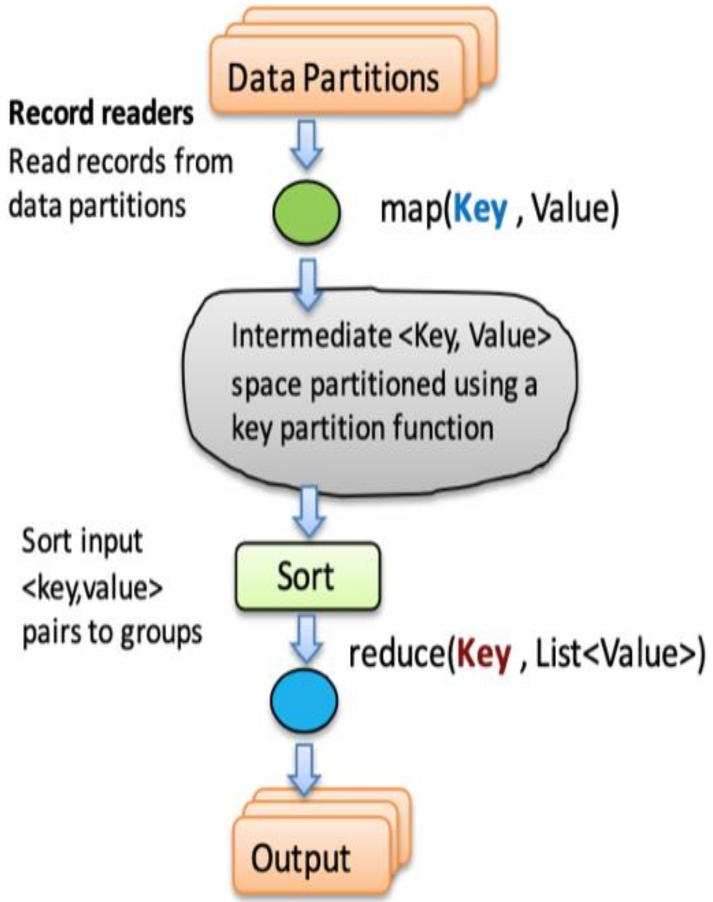
Input: a set of key/value pairs

User supplies two functions:

- map(k,v) → list(k1,v1)
- reduce(k1, list(v1)) → v2

(k1,v1) is an intermediate key/value pair

Output is the set of (k1,v2) pairs



When will you use this ?

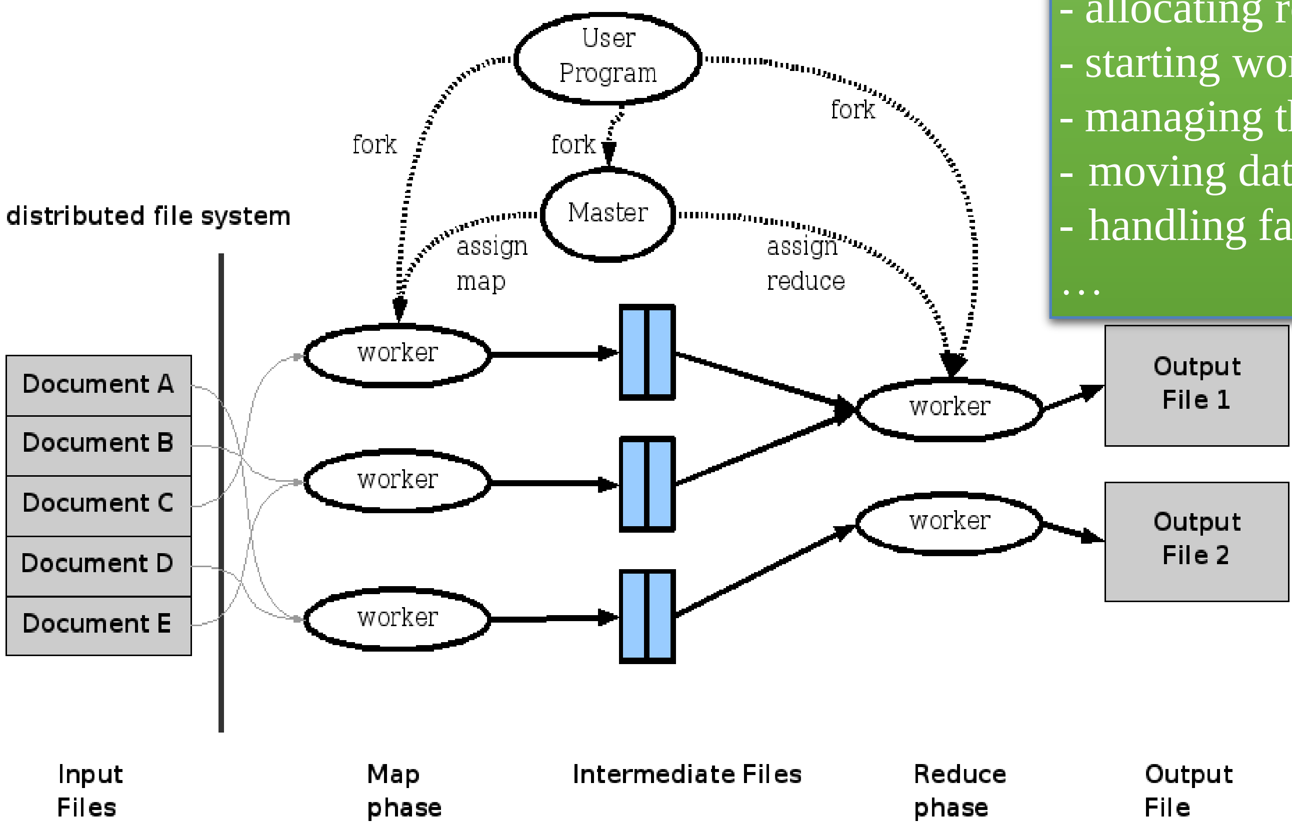
- Huge set of documents that don't fit into memory
  - So need file based processing in stages
- Lot of data partitioning (high data parallelism)
- Possibly simple merge among partitions (low cost inverse tree parallelism)

MapReduce: Execution overview

- Data centric design
- Move computation closer to data
- Intermediate results on disk
- Dynamic task scheduling

A MapReduce library and runtime does all the work for
- allocating resources,
- starting workers,
- managing them,
- moving data,
- handling failures

…



MapReduce origins

Created in Google on GFS

Open source version created as Apache Hadoop

Perform maps/reduces on data using many machines

- The system takes care of distributing the data and managing fault tolerance
- You just write code to map one element and reduce elements to a combined result

Separates how to do recursive divide-and-conquer from what computation to perform

- Old idea in higher-order functional programming transferred to large-scale distributed computing
- Complementary approach to database declarative queries
  - In SQL you don't actually write the low level query execution code
- Programmer needs to focus just on map and reduce logic and rest of the work is done by the map-reduce framework.
  - So **restricted programming interface** to the system to let the system do the distribution of work, job tracking, fault tolerance etc.

More complex example - sales data processing

	A	B	C	D	E	F	G	H	I	J	K	L
1	Transaction_date	Product	Price	Payment_	Name	City	State	Country	Account_Created	Last_Login	Latitude	Longitude
2	01-02-2009 06:17	Product1	1200	Mastercar	carolina	Basildon	England	United Kir	01-02-2009 06:00	01-02-2009 06:08	51.5	-1.11667
3	01-02-2009 04:53	Product1	1200	Visa	Betina	Parkville	MO	United Sta	01-02-2009 04:42	01-02-2009 07:49	39.195	-94.6819
4	01-02-2009 13:08	Product1	1200	Mastercar	Federica e	Astoria	OR	United Sta	01-01-2009 16:21	01-03-2009 12:32	46.18806	-123.83
5	01-03-2009 14:44	Product1	1200	Visa	Gouya	Echuca	Victoria	Australia	9/25/05 21:13	01-03-2009 14:22	-36.1333	144.75
6	01-04-2009 12:56	Product2	3600	Visa	Gerd W	Cahaba He	AL	United Sta	11/15/08 15:47	01-04-2009 12:45	33.52056	-86.8025
7	01-04-2009 13:19	Product1	1200	Visa	LAURENCE	Mickleton	NJ	United Sta	9/24/08 15:19	01-04-2009 13:04	39.79	-75.2381
8	01-04-2009 20:11	Product1	1200	Mastercar	Fleur	Peoria	IL	United Sta	01-03-2009 09:38	01-04-2009 19:45	40.69361	-89.5889
9	01-02-2009 20:09	Product1	1200	Mastercar	adam	Martin	TN	United Sta	01-02-2009 17:43	01-04-2009 20:01	36.34333	-88.8503
10	01-04-2009 13:17	Product1	1200	Mastercar	Renee Elis	Tel Aviv	Tel Aviv	Israel	01-04-2009 13:03	01-04-2009 22:10	32.06667	34.76667
11	01-04-2009 14:11	Product1	1200	Visa	Aidan	Chatou	Ile-de-Fra	France	06-03-2008 04:22	01-05-2009 01:17	48.88333	2.15
12	01-05-2009 02:42	Product1	1200	Diners	Stacy	New York	NY	United Sta	01-05-2009 02:23	01-05-2009 04:59	40.71417	-74.0064
13	01-05-2009 05:39	Product1	1200	Amex	Heidi	Eindhover	Noord-Bra	Netherlan	01-05-2009 04:55	01-05-2009 08:15	51.45	5.466667
14	01-02-2009 09:16	Product1	1200	Mastercar	Sean	Shavano P	TX	United Sta	01-02-2009 08:32	01-05-2009 09:05	29.42389	-98.4933
15	01-05-2009 10:08	Product1	1200	Visa	Georgia	Eagle	ID	United Sta	11-11-2008 15:53	01-05-2009 10:05	43.69556	-116.353
16	01-02-2009 14:18	Product1	1200	Visa	Richard	Riverside	NJ	United Sta	12-09-2008 12:07	01-05-2009 11:01	40.03222	-74.9578
17	01-04-2009 01:05	Product1	1200	Diners	Leanne	Julianstov	Meath	Ireland	01-04-2009 00:00	01-05-2009 13:36	53.67722	-6.31917

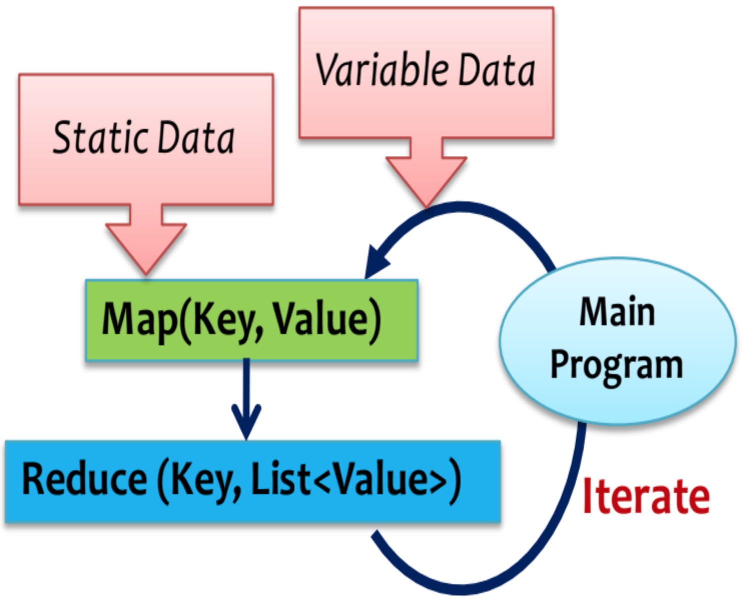
```
Argentina       1
Australia       38
Austria 7
Bahrain 1
Belgium 8
Bermuda 1
Brazil  5
Bulgaria        1
CO      1
Canada  76
Cayman Isls     1
China   1
Costa Rica      1
Country 1
Czech Republic  3
Denmark 15
Dominican Republic      1
Finland 2
France  27
Germany 25
Greece  1
Guatemala       1
Hong Kong       1
Hungary 3
Iceland 1
India   2
```

count tx by country

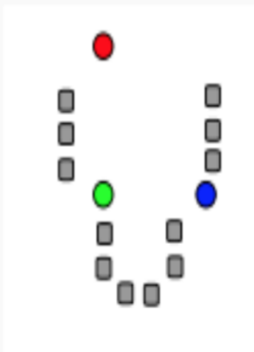
https://www.guru99.com/create-your-first-hadoop-program.html

Iterative Map Reduce

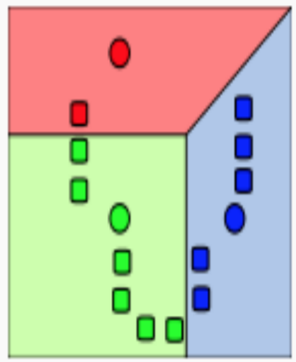
- MapReduce is a one-pass computation
- Many applications, esp in ML and Data Mining areas, need to iteratively process data
- So they need iterative execution of map reduce jobs
- An approach is to create a main program that calls the core map reduce with variable data
- Core program also checks for convergence
  - error bound (e.g. k-means clustering)
  - fixed iterations



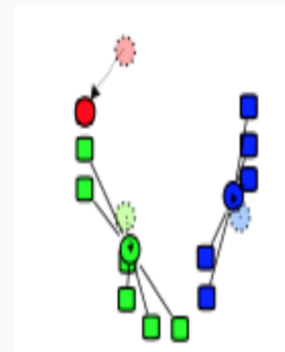
Example 1: K-means clustering



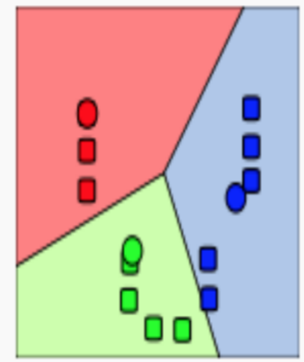
1) *k* initial "means" (in this case *k*=3) are randomly selected from the data set (shown in color).



2) *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

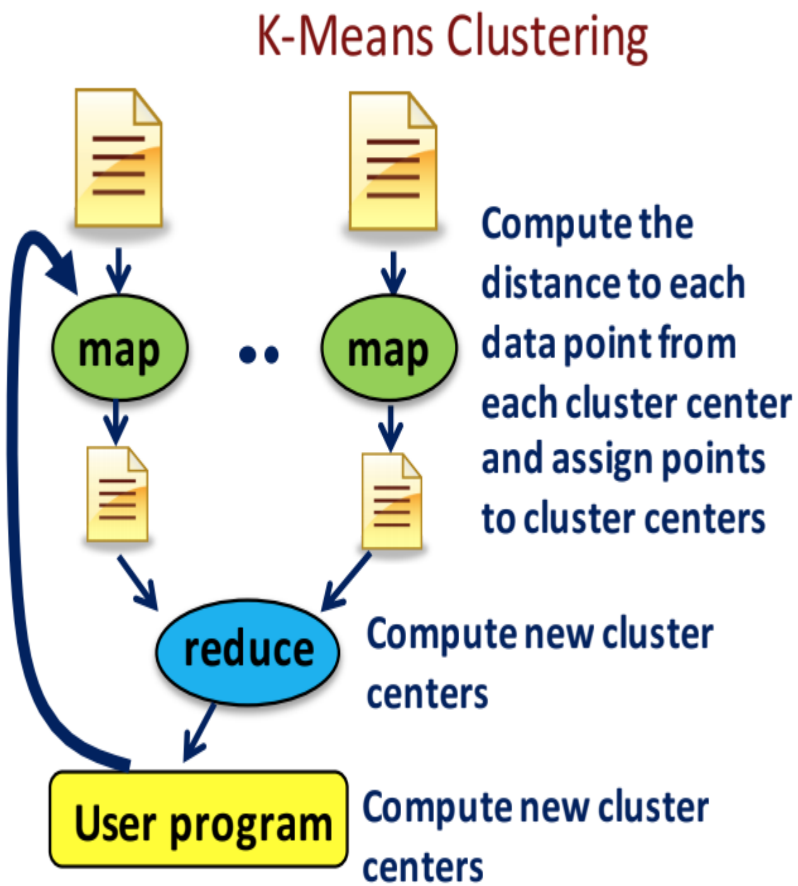


3) The centroid of each of the *k* clusters becomes the new means.



4) Steps 2 and 3 are repeated until convergence has been reached.

K-means as iterative map reduce



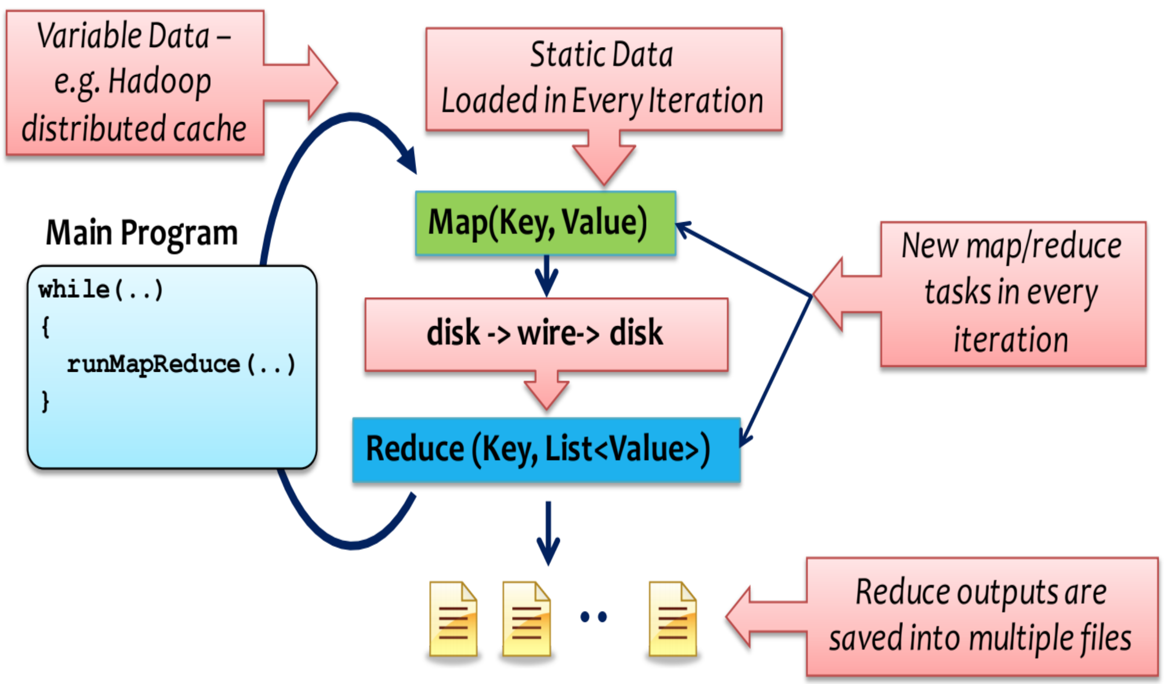
- The MapReduce program driver is responsible for repeating the steps via an iterative construct.
- Within each iteration map and reduce steps are called.
- Each map step reuses the result produced in previous reduce step.
  - e.g. k centers computed

https://github.com/thomasjungblut/mapreduce-kmeans/tree/master/src/de/jungblut/clustering/mapreduce

Iterations using existing runtimes

Loop implemented on top of existing file-based single step map-reduce core

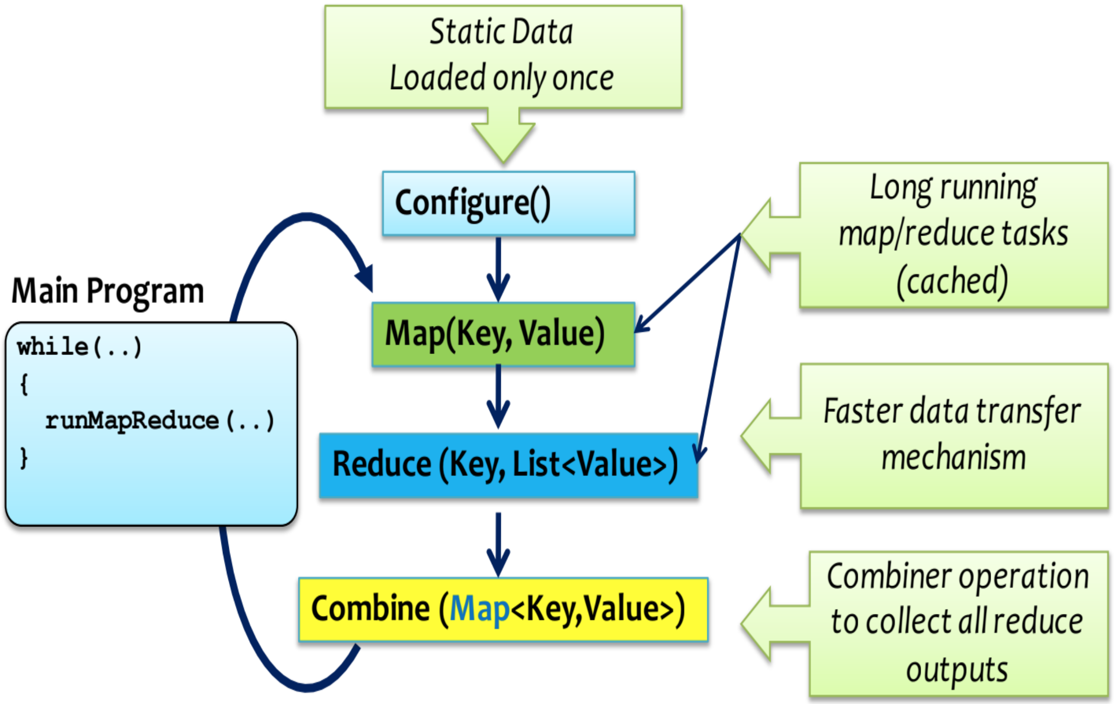
Large overheads from
- re-initialization of tasks
- reloading of static data
- communication and data transfers



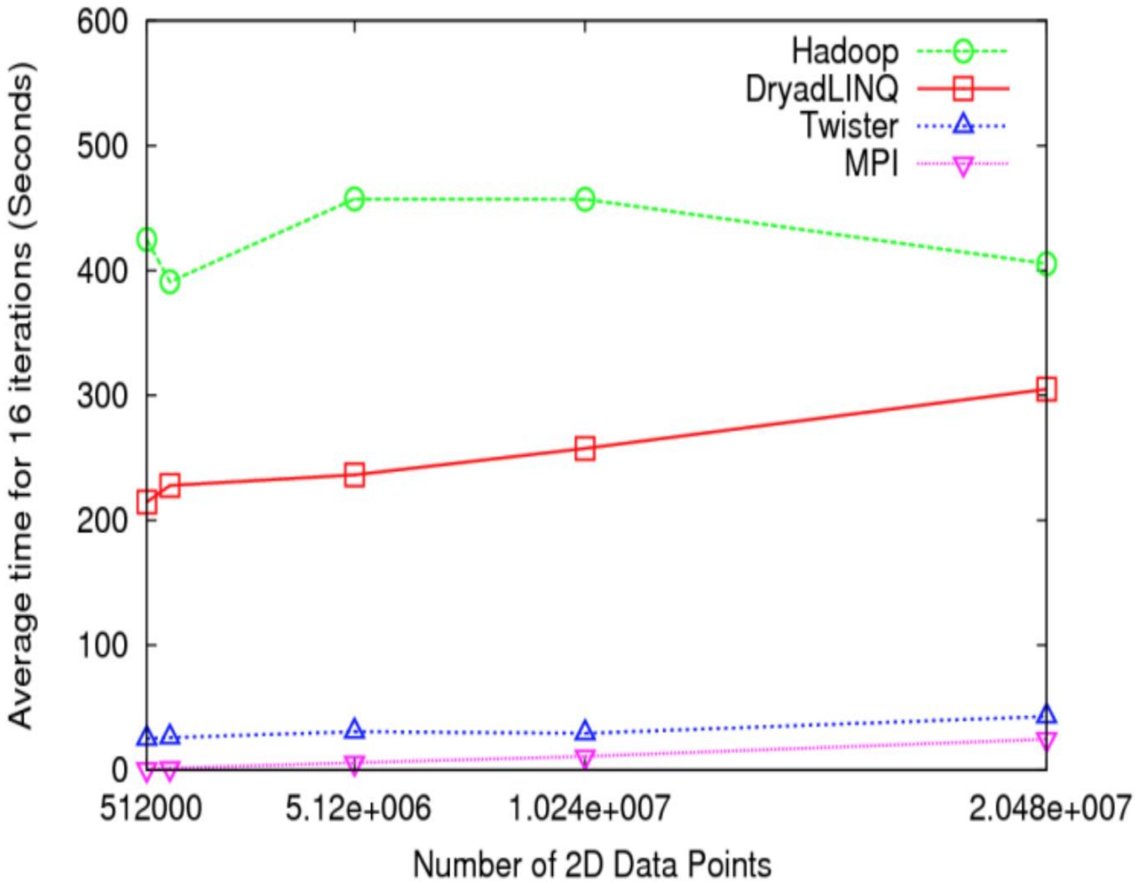
DistributedCache: https://hadoop.apache.org/docs/r2.6.3/api/org/apache/hadoop/filecache/DistributedCache.html

MapReduce++ : Iterative MapReduce

- Some optimizations are done on top of existing model
  - Static data loaded once
  - Cached tasks across invocations
  - Combine operations



The optimisations indeed help



K-means clustering using various programming models

Iterative MapReduce: Other options

HaLoop

- Modifies Hadoop scheduling to make it loop aware
- Implements caches to avoid going to disk between iterations
- Optional reading: Paper in Proceedings of the VLDB Endowment 3(1):285-296, Sep 2010

Spark

- Uses in-memory computing to speed up iterations
- An in-memory structure called RDD : Resilient Distributed Dataset replaces files on disk
- Ideal for iterative computations that reuse lot of data in each iteration