BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DSECL ZG 522: Big Data Systems
Session 7 - Distributed Programming

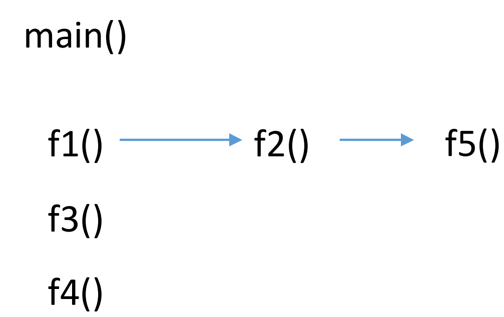
Janardhanan PS
janardhanan.ps@wilp.bits-pilani.ac.in

Topics for today

- **Top down design**
- Types of parallelism
- MapReduce programming model
- See how a map reduce program works on Hadoop
- Iterative MapReduce
- Hands on demo of K-Means clustering using iterative MapReduce

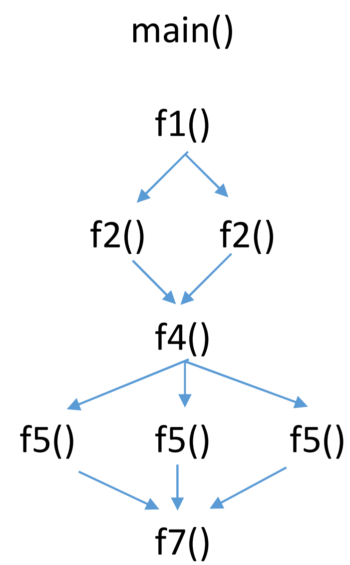
Top down design - Sequential context

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



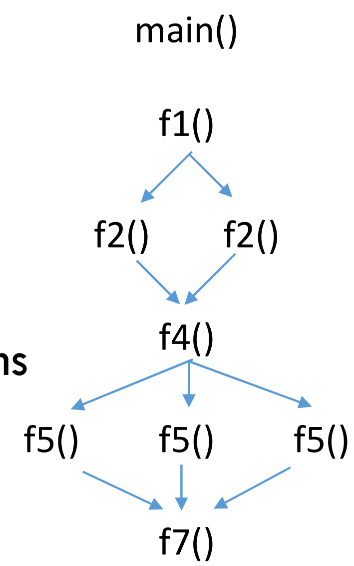
Top down design – Parallel context

- In the context of a parallel program, 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
    - Utilise all available processors
  - 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

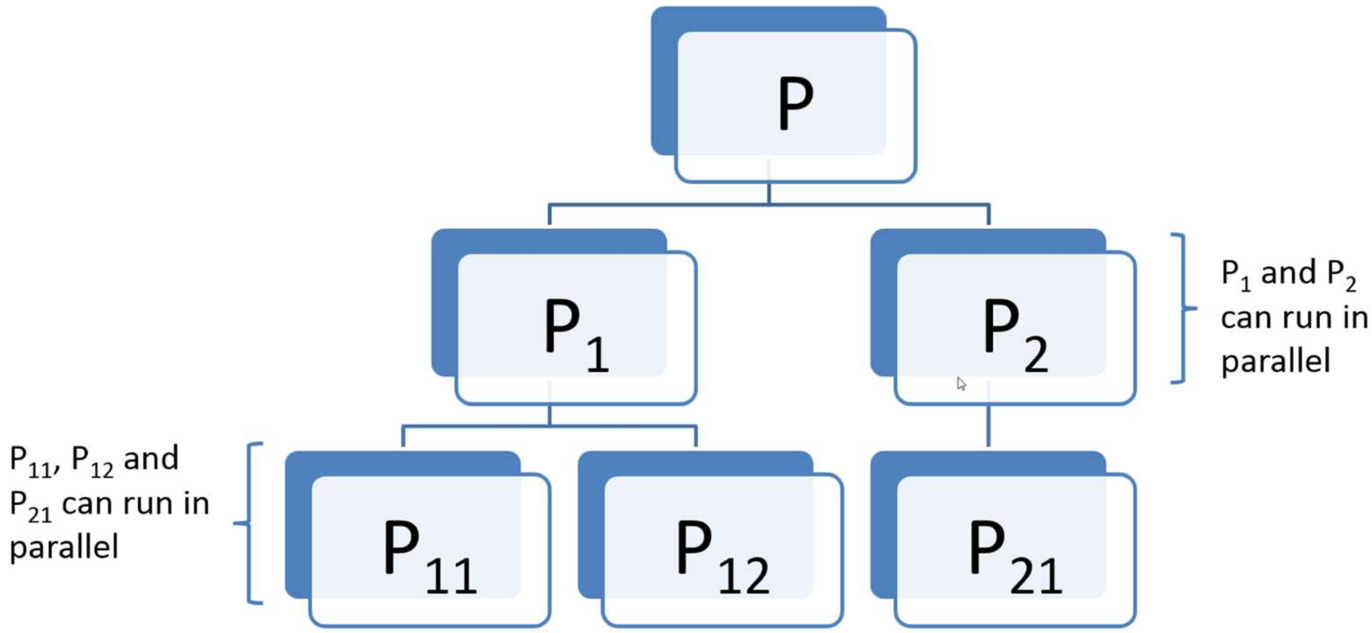
- In conventional top-down design for sequential systems
  - Keep number of sub-problems manageable
  - Because need to keep track of them as computation progresses
- In parallel systems, it is dictated by number of processors
  - Processor utilization is the key
  - If there are N processors, we can potentially have N sub-problems



How many processors ?

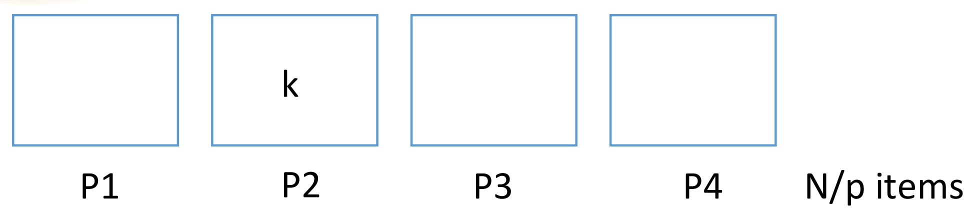
Top-down design

- At each level, problems need to run in parallel



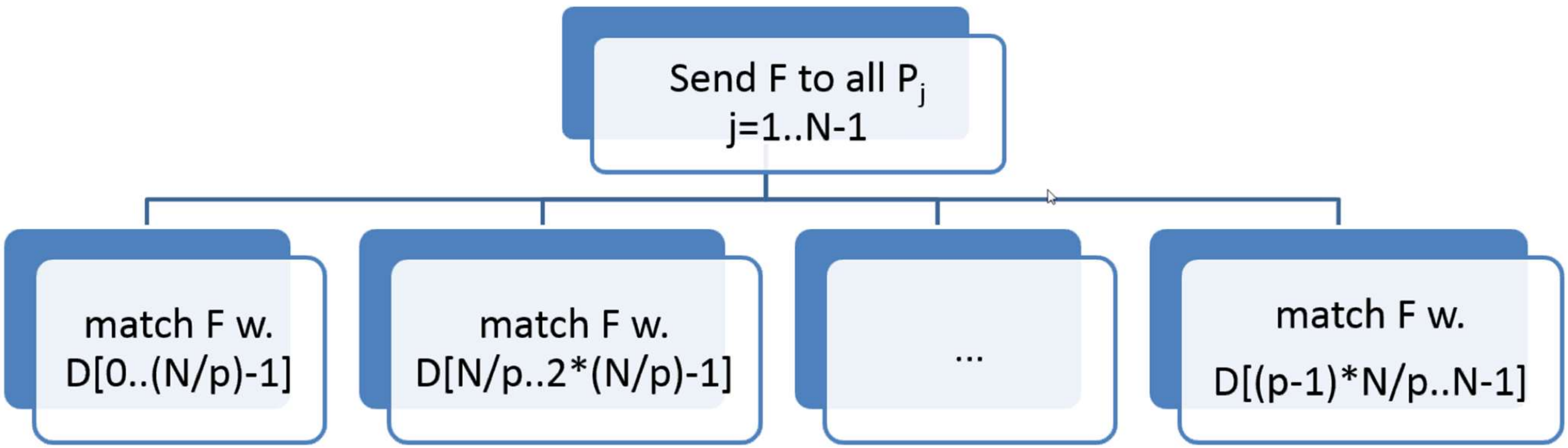
Example 1 - Keyword search in list

- 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 i^{th} processor has found key in j^{th} position
  - Combination: One or more positions are collected at processor 0
- Speedup: p
- Time complexity: O(Log(N/p))



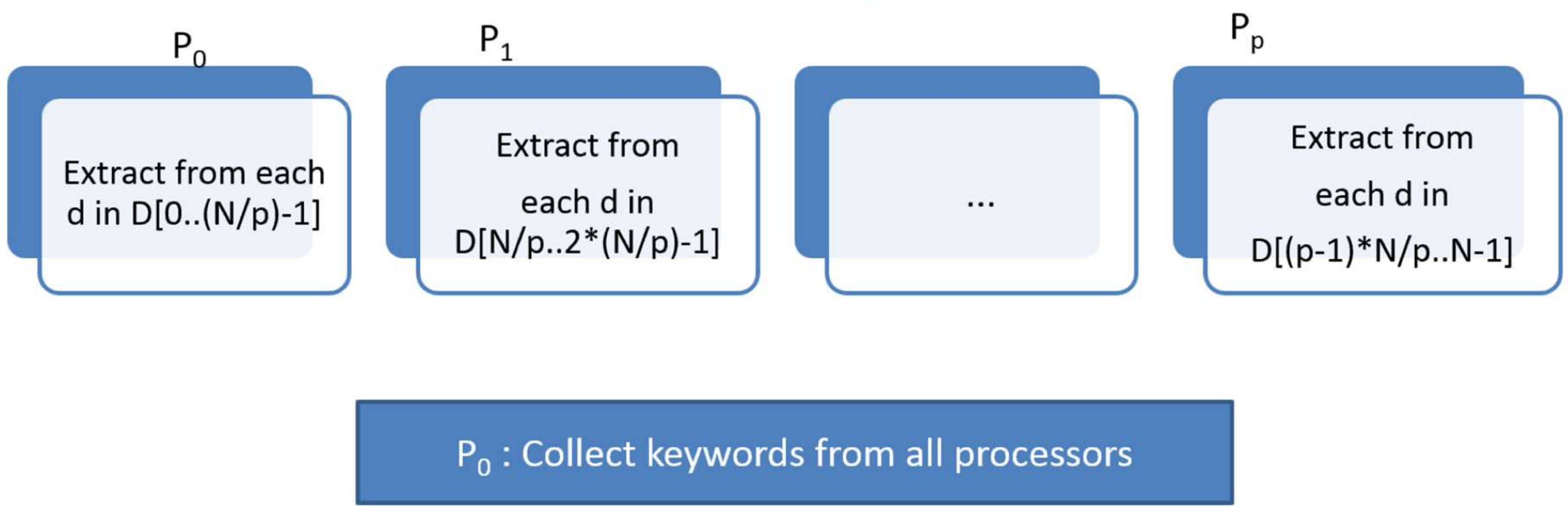
Example 2 - Fingerprint matching

- 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



Topics for today

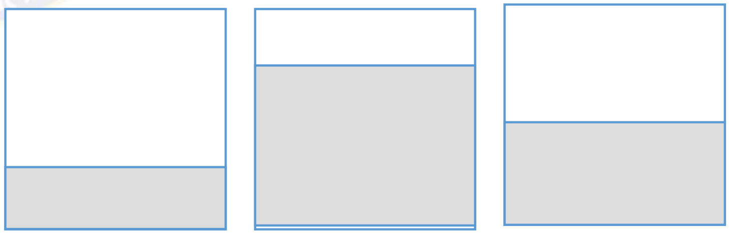
- Top-down design
- **Types of parallelism**
- MapReduce programming model
- See how a map reduce program works using Hadoop
- Iterative MapReduce

Types of Parallelism

1. Data Parallelism
2. Tree Parallelism
3. Task Parallelism
4. Request Parallelism

1. 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 overhead is minimal
  - Unbalanced data size / work or heterogenous nodes will lead to higher execution time



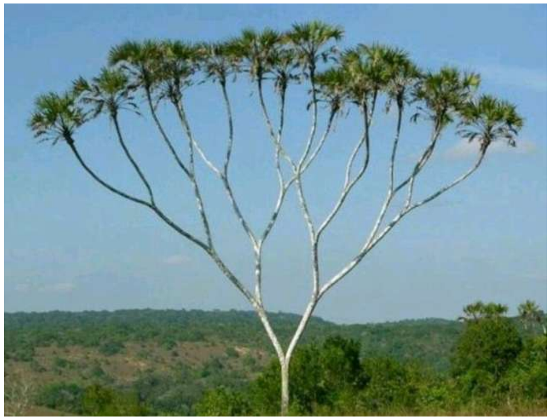
Where data parallelism is not possible

- 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
  - Time Complexity
    - Best case - $O(n.\log(n))$
    - Worst case – $O(n^2)$

[Sorting (Bubble, Selection, Insertion, Merge, Quick, Counting, Radix) - VisuAlgo]

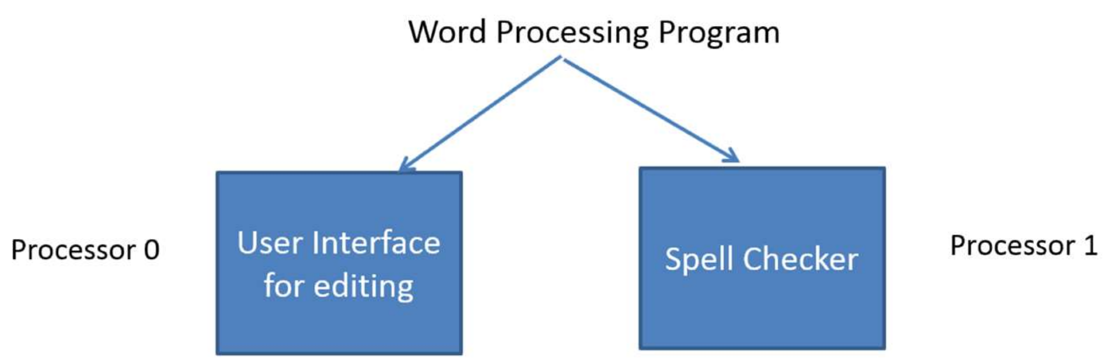
2. Tree parallelism

- Dynamic version of divide and conquer - partitions are done dynamically
- Division of problem into sub-problems happens at execution time
  - Sub-problem is identical in structure to the larger problem
  - What is the division size ?



3. Task parallelism - Example 1 : Word processor

- Parallel tasks that work on the same data
  - Unlike data and tree parallelism, 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 statistics on Ls
  - Run tasks in parallel

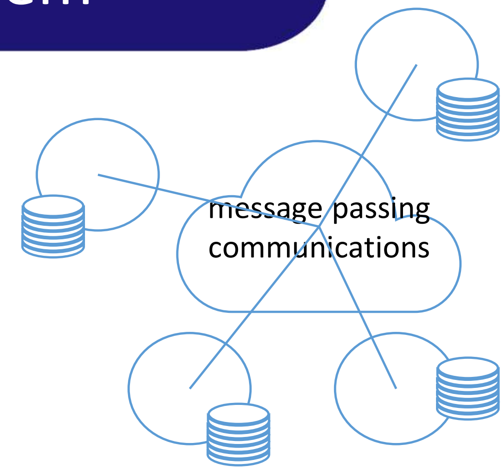
Task parallelism summary

- Identify sub-tasks based on functionality with no common function
  - In Tree and Data parallel, the tasks are identical functions
- 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 BigData

4. 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
  - Microservices
  - Socket programming
- 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

Topics for today

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

MapReduce origins

- Created in Google to run on GFS
- Open-source version created as Apache Hadoop
- Perform maps/reduces on data using many machines
- The framework 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.

MapReduce Evolution

- We have a large text file of words, one word in a line
- We need to count the number of times each distinct word appears in the file
- Sample application: analyze web server logs to find popular URLs

- Word Count Solution Design Options:

  1: Entire file fits in memory

  - Read file into memory and count

  2: File too large for memory, but all <word, count> pairs fit in memory

  - Read file line by line and maintain words and counts in memory

  3: File on disk, too many distinct words to fit in memory

```
sort words.txt | uniq -c
```

Solution for multiple large files on disk

To make it slightly harder, suppose we have a large corpus of documents

Count the number of times each distinct word occurs in the corpus

- `words(docs/*) | sort | uniq -c`

where **`words`** takes a file in a folder and outputs the words in it, one to a line

- The above captures the essence of MapReduce
- Great thing is it is naturally parallelizable

MapReduce is conceptually like a UNIX pipeline

- One function (Map) processes data
- Output is input to another function (Reduce)

cat words.txt | sort | uniq -c | cat > file

input | **map** | shuffle | **reduce** | output

→ Developer specifies two functions:

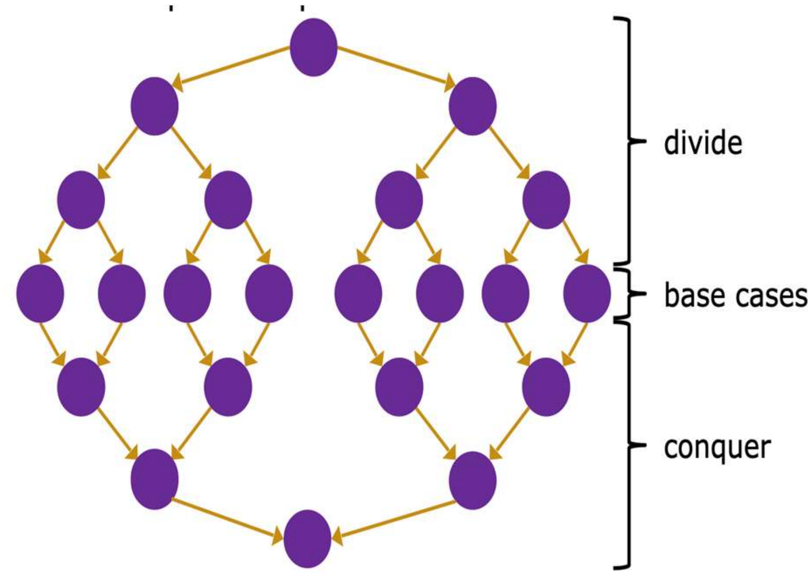
- map() - User code
- reduce() - User code

→ Rest of the job is done by the MapReduce framework

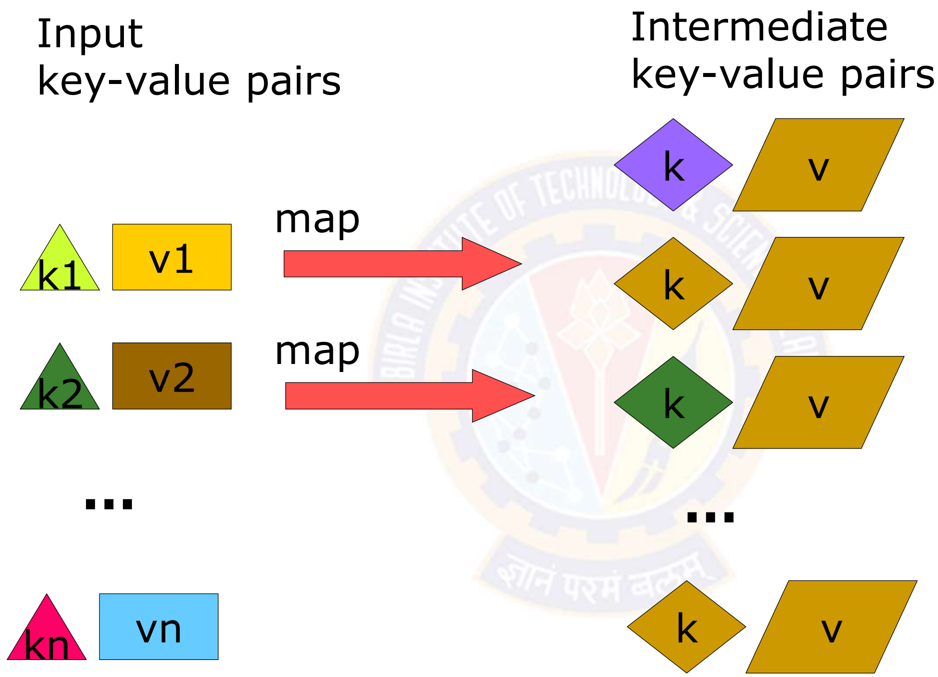
→ Tune the configuration parameters of the MapReduce framework for performance

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



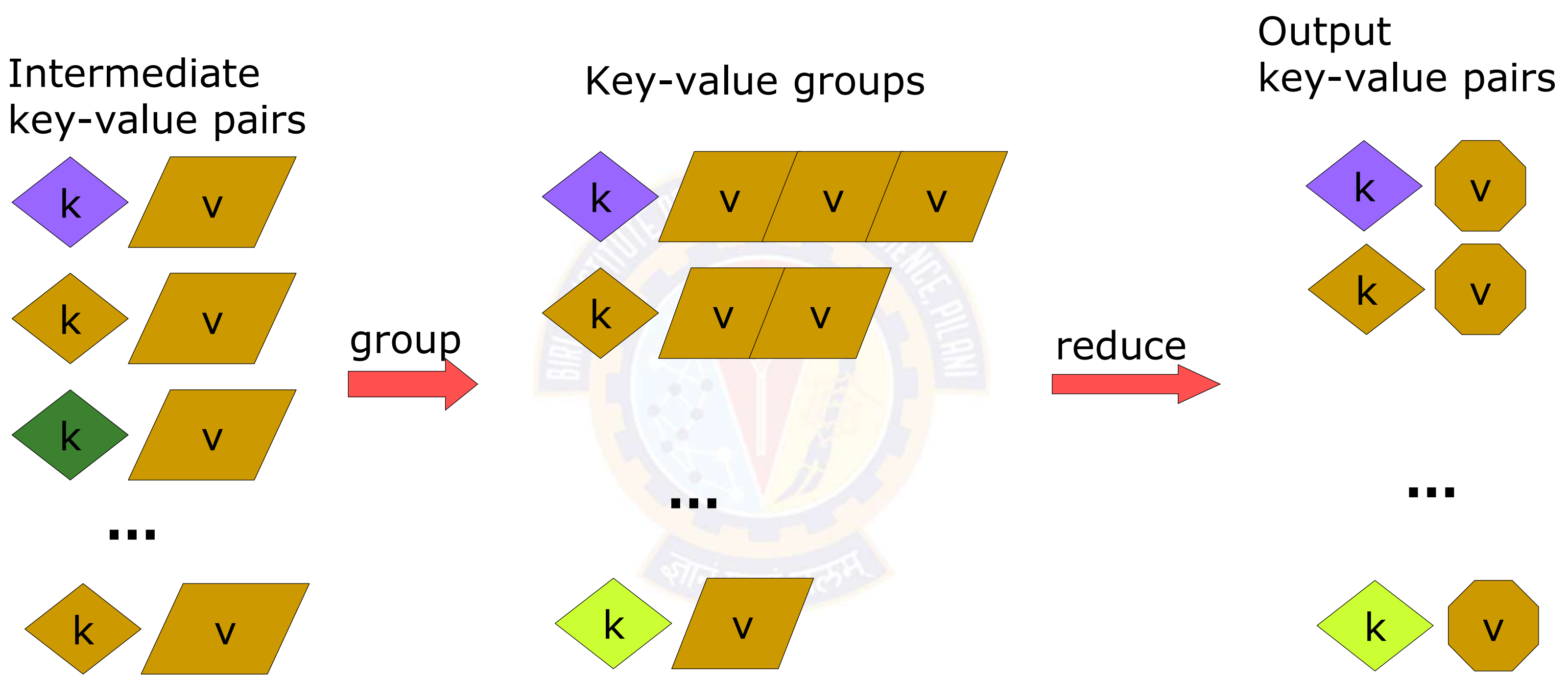
MapReduce: The Map Step



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

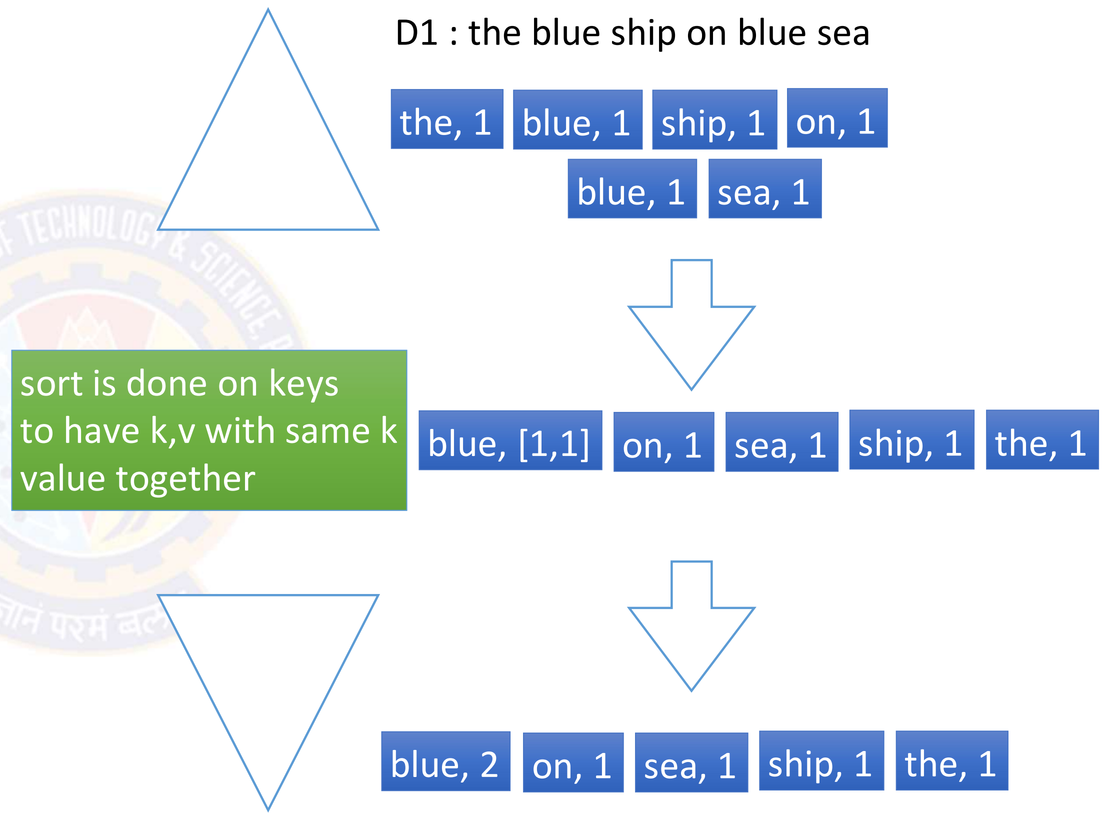
(word, final-count)
~ SQL aggregation

Adapted from Jeff Ullman's course slides

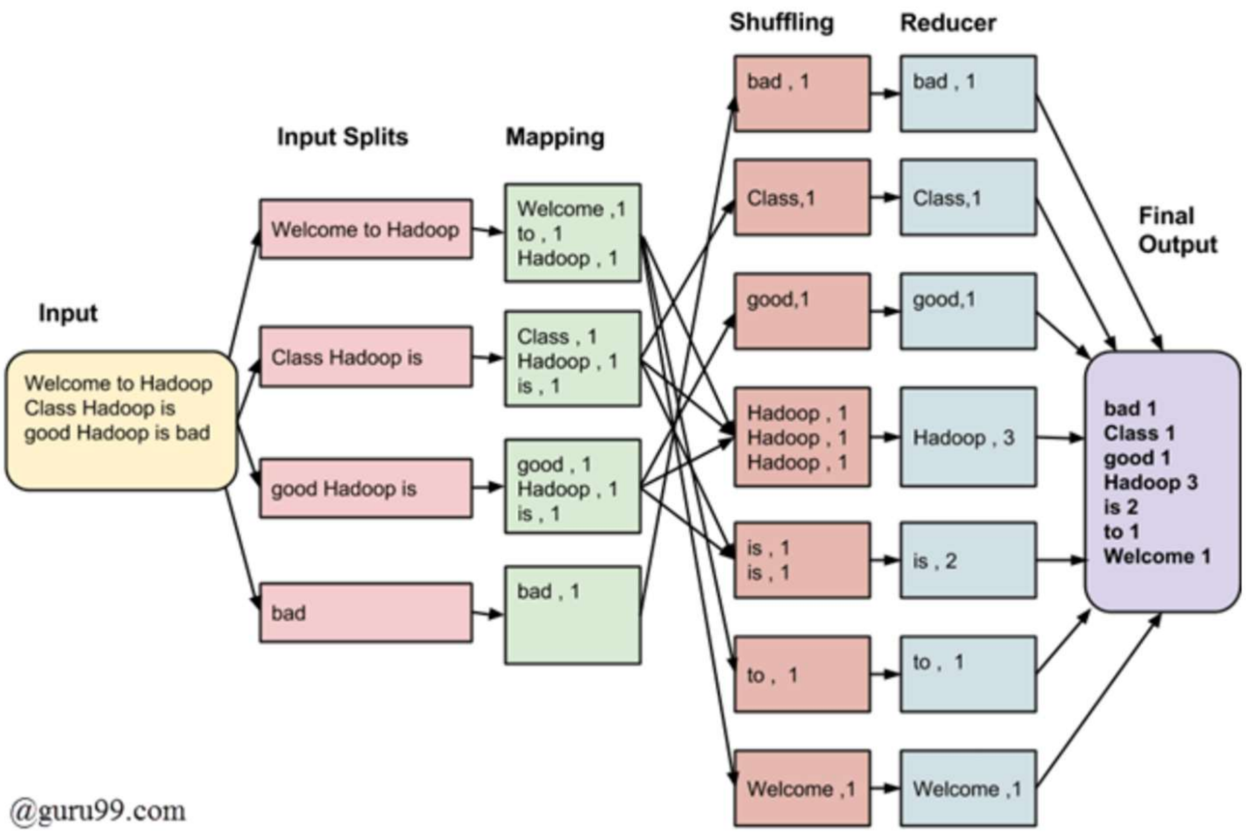
Example: Word Count using MapReduce (Pseudo code)

```
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)
```



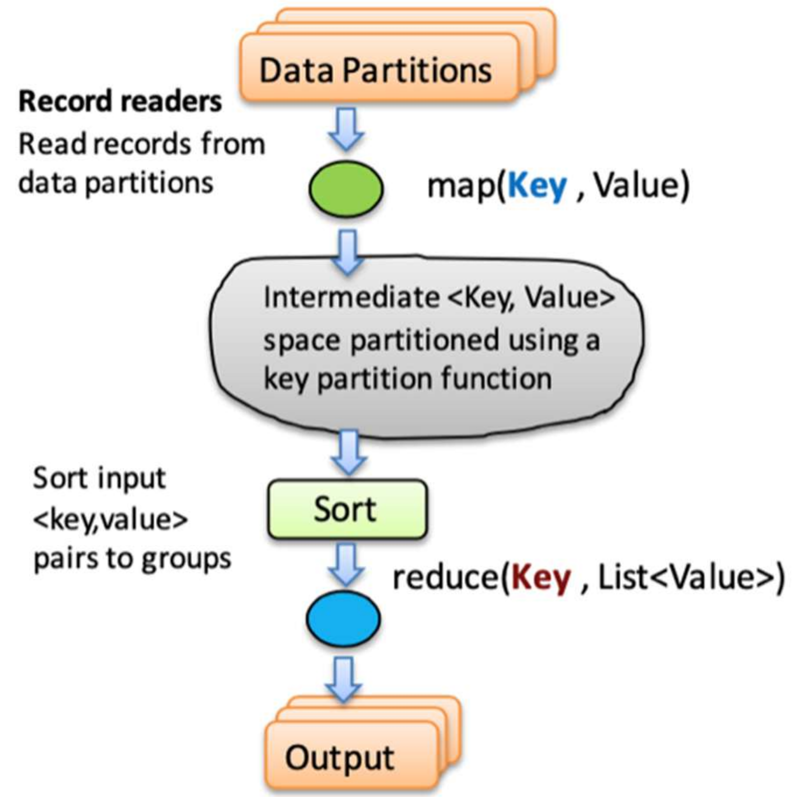
Word count (2)



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

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 ?

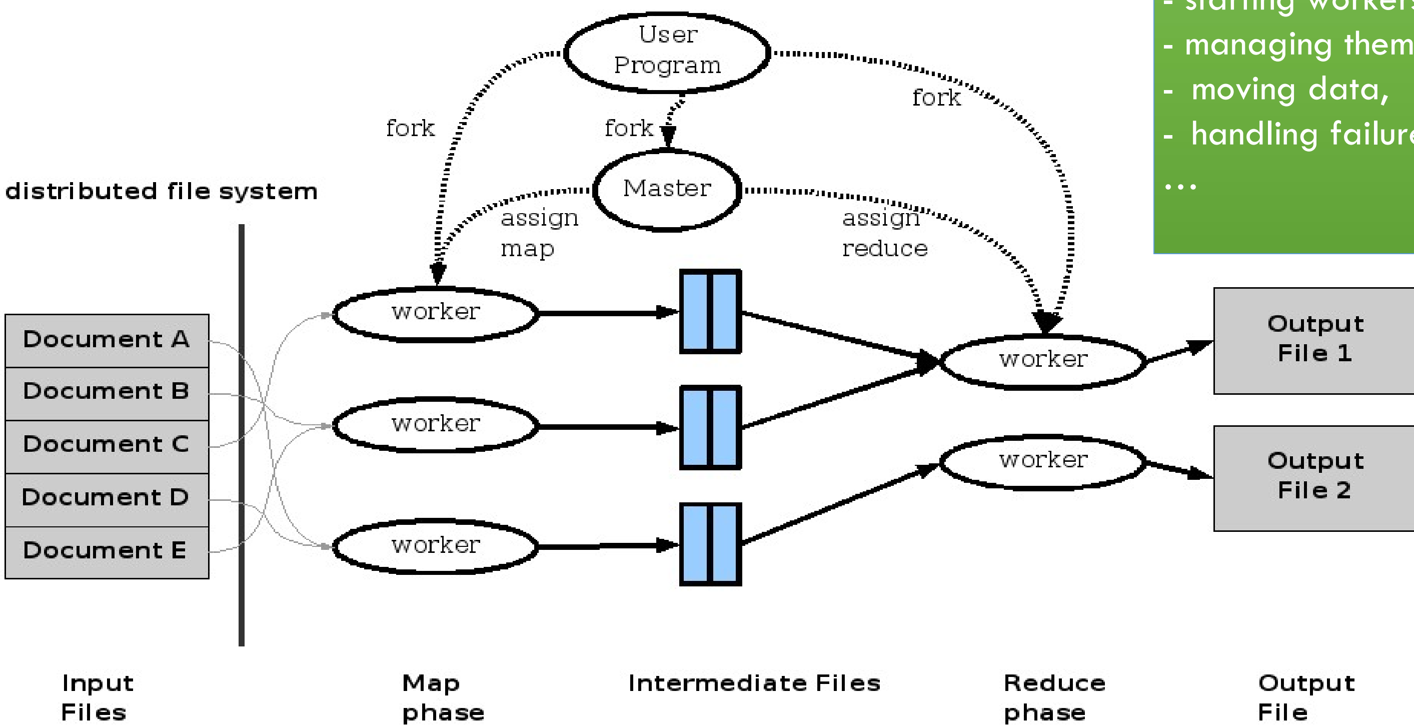
- Huge set of documents that don’t fit into memory
  - So, need file-based processing in stages, e.g. Hadoop
  - But can also do this in memory - e.g. Spark
- Lot of data partitioning (high data parallelism)
- Possibly simple merge among partitions (low cost inverse tree parallelism)

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

…



Topics for today

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

MapReduce example - sales data processing

Sales by country

	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	Netherlar	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

Unravelling a MapReduce program - Driver (1)

```
package SalesCountry;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class SalesCountryDriver {

  public static void main(String[] args) {
    JobClient my_client = new JobClient();
    // Create a configuration object for the job
    JobConf job_conf = new
    JobConf(SalesCountryDriver.class);

    // Set a name of the Job
    job_conf.setJobName("SalePerCountry");

    // Specify data type of output key and value
    job_conf.setOutputKeyClass(Text.class);
    job_conf.setOutputValueClass(IntWritable.class);

     . . .
```

package name of jar

hadoop libs

driver class that contains main()

Hadoop job with driver class with SalesPerCountry as the application name

Output data types defined based on existing hadoop classes

Unravelling a MapReduce program - Driver (2)

```
      . . .

    // Specify names of Mapper and Reducer Class
    job_conf.setMapperClass(SalesCountry.SalesMapper.class);
    job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);

    // Set input and output directories
    // arg[0] = name of input directory on HDFS, and
    // arg[1] =  name of output directory to be created
    // to store the output file.
    FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

    my_client.setConf(job_conf);
    try {
      // Run the job
      JobClient.runJob(job_conf);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Mapper and Reducer classes in the jar pkg

Paths to read input and send output

Send job for execution

Unravelling a MapReduce program - Mapper

```
package SalesCountry;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class SalesMapper extends MapReduceBase implements Mapper <LongWritable, Text,
Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
// can add some data structure here for across map() instances
        public void map(LongWritable key, Text value, OutputCollector <Text, IntWritable> output,
        Reporter reporter) throws IOException {

                String valueString = value.toString();
                String[] SingleCountryData = valueString.split(",");
                output.collect(new Text(SingleCountryData[7]), one);
        }
}
```

app jar package

hadoop libs

map function w inputs
- one or more <key, value> pairs
- output collector
- ...

Map logic

<India, 1> <USA, 1> <India, 1> ....

Unravelling a MapReduce program - Reducer

```
package SalesCountry;
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class SalesCountryReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text,
IntWritable> {
    // can add some data structure here for across reduce() instances
     public void reduce(Text t_key, Iterator<IntWritable> values, OutputCollector<Text,IntWritable> output,
       Reporter reporter) throws IOException {
            Text key = t_key;
            int frequencyForCountry = 0;
            while (values.hasNext()) {
                   // replace type of value with the actual type of our value
                   IntWritable value = (IntWritable) values.next();
                   frequencyForCountry += value.get();

            }
            output.collect(key, new IntWritable(frequencyForCountry));
     }
}
```

Pkg and includes

reduce function w inputs
- key and value list, e.g. <India, {1,1,1,1}>
- output collector
- …

Reduce logic
- calculate frequency

For each reduce task, 1 output file created
Can control #reducer in driver

Running and checking status

```
[root@centos-s-4vcpu-8gb-blr1-01 source]# jps
7744 SecondaryNameNode
8357 ResourceManager
8581 NodeManager
13541 Jps
7238 DataNode
10428 MRAppMaster
6910 NameNode
```

```
[root@centos-s-4vcpu-8gb-blr1-01 source]# ls -l
total 148
-rw-r--r--. 1 root root     44 Apr 18 01:11 Manifest.txt
-rw-r--r--. 1 root root   2966 Apr 18 01:12 ProductSalePerCountry.jar
drwxr-xr-x. 2 root root     96 Apr 19 00:57 SalesCountry
-rw-r--r--. 1 root root   1529 Apr 18 00:57 SalesCountryDriver.java
-rw-r--r--. 1 root root    746 Apr 18 00:56 SalesCountryReducer.java
-rw-r--r--. 1 root root 123637 Jun  6 16:56 SalesJan2009.csv
-rw-r--r--. 1 root root    661 Apr 18 00:56 SalesMapper.java
drwxr-xr-x. 3 root root    157 Jun  6 16:53 units
-rw-r--r--. 1 root root    219 Apr 18 21:10 units.csv
[root@centos-s-4vcpu-8gb-blr1-01 source]# hadoop jar ProductSalePerCountry.jar /SalesJan2009.csv /output
21/06/06 17:27:37 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applic
21/06/06 17:27:38 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/06 17:27:39 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/06/06 17:27:39 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execut
remedy this.
21/06/06 17:27:40 INFO mapred.FileInputFormat: Total input files to process : 1
21/06/06 17:27:40 INFO mapreduce.JobSubmitter: number of splits:2
21/06/06 17:27:40 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1622978356181_0002
21/06/06 17:27:41 INFO conf.Configuration: resource-types.xml not found
21/06/06 17:27:41 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
21/06/06 17:27:41 INFO resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
21/06/06 17:27:41 INFO resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
21/06/06 17:27:41 INFO impl.YarnClientImpl: Submitted application application_1622978356181_0002
21/06/06 17:27:41 INFO mapreduce.Job: The url to track the job: http://centos-s-4vcpu-8gb-blr1-01:8088/proxy/application_1622978356181_0002/
21/06/06 17:27:41 INFO mapreduce.Job: Running job: job_1622978356181_0002
21/06/06 17:27:49 INFO mapreduce.Job: Job job_1622978356181_0002 running in uber mode : false
21/06/06 17:27:49 INFO mapreduce.Job:  map 0% reduce 0%
21/06/06 17:28:03 INFO mapreduce.Job:  map 100% reduce 0%
```

Logged in as: dr.who

MapReduce Application application_1622978356181_0002

- Cluster
- Application
  - About
  - Jobs
- Tools

Active Jobs

Show 20 entries

Search:

Job ID	Name	State	Map Progress	Maps Total	Maps Completed	Reduce Progress	Reduces Total	Reduces Completed
job_1622978356181_0002	SalePerCountry	RUNNING		2	2		1	0

Showing 1 to 1 of 1 entries

First Previous 1 Next Last

MapReduce stats

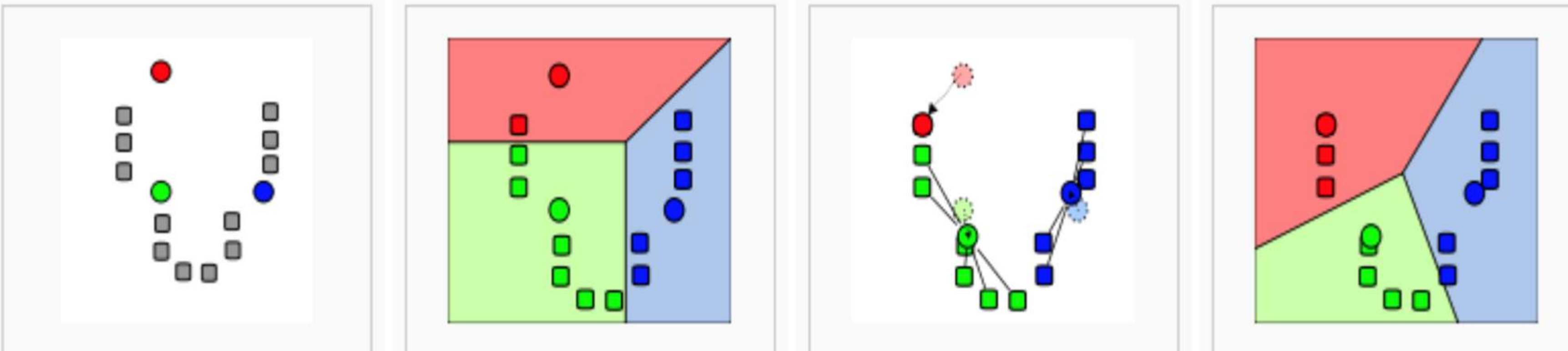
```
File System Counters
        FILE: Number of bytes read=17747
        FILE: Number of bytes written=660936
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=127535
        HDFS: Number of bytes written=661
        HDFS: Number of read operations=9
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
Job Counters
        Launched map tasks=2
        Launched reduce tasks=1
        Data-local map tasks=2
        Total time spent by all maps in occupied slots (ms)=14658
        Total time spent by all reduces in occupied slots (ms)=7011
        Total time spent by all map tasks (ms)=14658
        Total time spent by all reduce tasks (ms)=7011
        Total vcore-milliseconds taken by all map tasks=14658
        Total vcore-milliseconds taken by all reduce tasks=7011
        Total megabyte-milliseconds taken by all map tasks=15009792
        Total megabyte-milliseconds taken by all reduce tasks=7179264
```

```
Map-Reduce Framework
        Map input records=999
        Map output records=999
        Map output bytes=15743
        Map output materialized bytes=17753
        Input split bytes=180
        Combine input records=0
        Combine output records=0
        Reduce input groups=58
        Reduce shuffle bytes=17753
        Reduce input records=999
        Reduce output records=58
        Spilled Records=1998
        Shuffled Maps =2
        Failed Shuffles=0
        Merged Map outputs=2
        GC time elapsed (ms)=585
        CPU time spent (ms)=2510
        Physical memory (bytes) snapshot=772923392
        Virtual memory (bytes) snapshot=6393163776
        Total committed heap usage (bytes)=527433728
Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
File Input Format Counters
        Bytes Read=127355
File Output Format Counters
        Bytes Written=661
```

Topics for today

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

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.

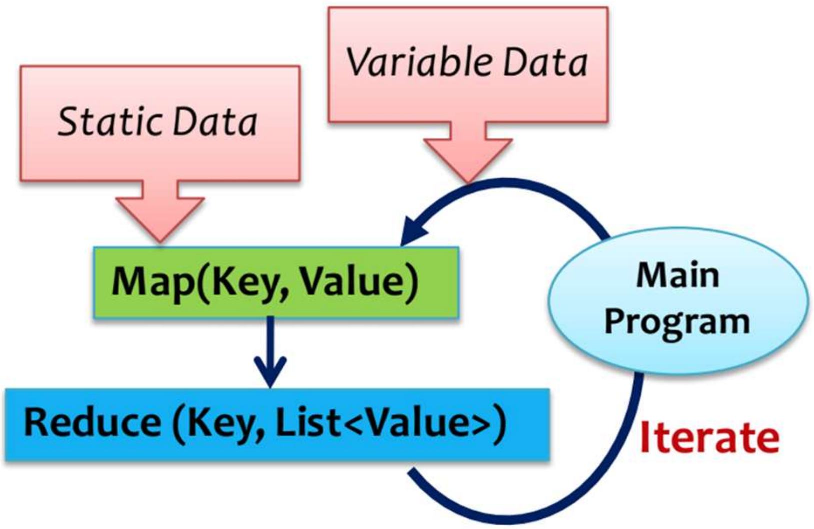
3) The centroid of each of the *k* clusters becomes the new means.

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

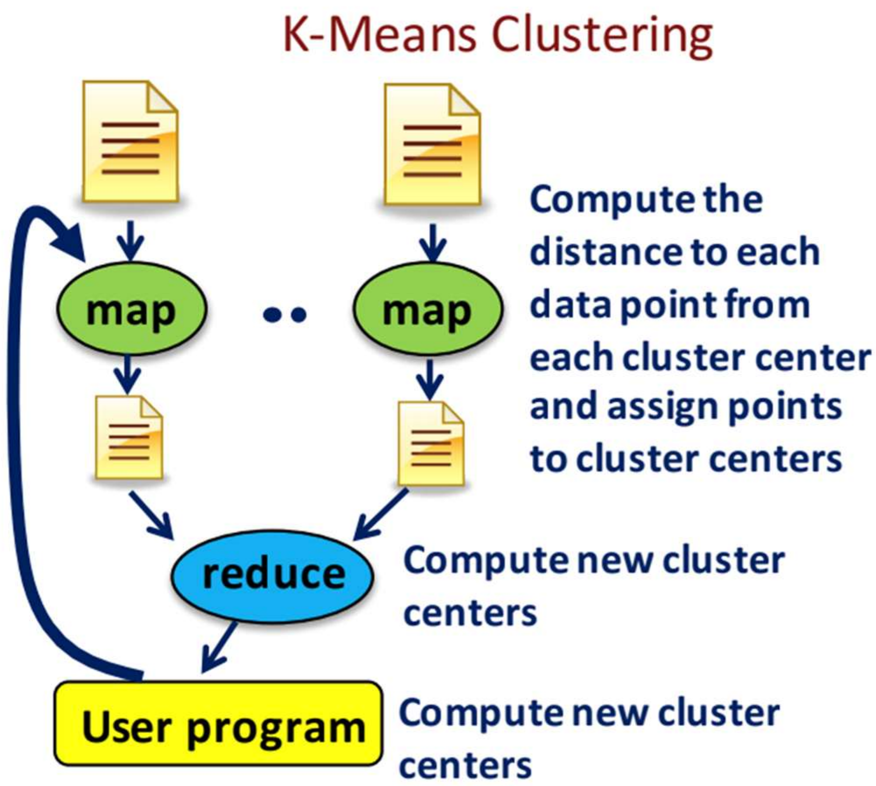
http://shabal.in/visuals/kmeans/2.html

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



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

K-Means Clustering by Iterative MapReduce

Hands on demo

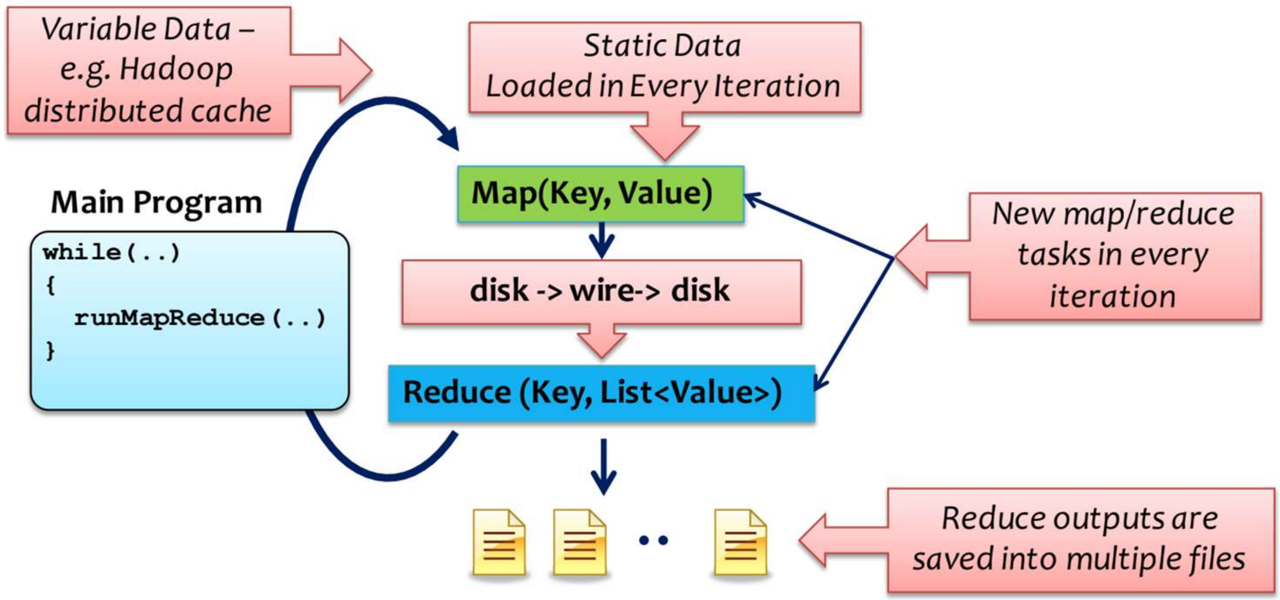
- 20Newsgroups folder - a set of around 20,000 postings to 20 different newsgroups with 1000 postings
- Convert all the newsgroup postings and turn them into bag-of-words vectors. – /data folder in HDFS
- Run a program that will choose an initial set of cluster centroids from the data – /clusters folder in HDFS (randomly sampled from the vectors)
- Run KMeans on Hadoop - hadoop jar MapRedKMeans.jar KMeans /data /clusters 3
  This will run 3 iterations of the KMeans algorithm on top of all documents in the 20_newsgroups data set. This means that three separate MapReduce jobs will be run in sequence.
- The centroids produced at the end of:

1. Iteration 1 will be put into the HDFS directory "/clusters1",
2. Iteration 2 will be put into the HDFS directory "/clusters2",
3. Iteration 3 will be put into the HDFS directory "/clusters3",

Reference: https://cmj4.web.rice.edu/MapRedKMeans.html

Iterations using existing runtimes

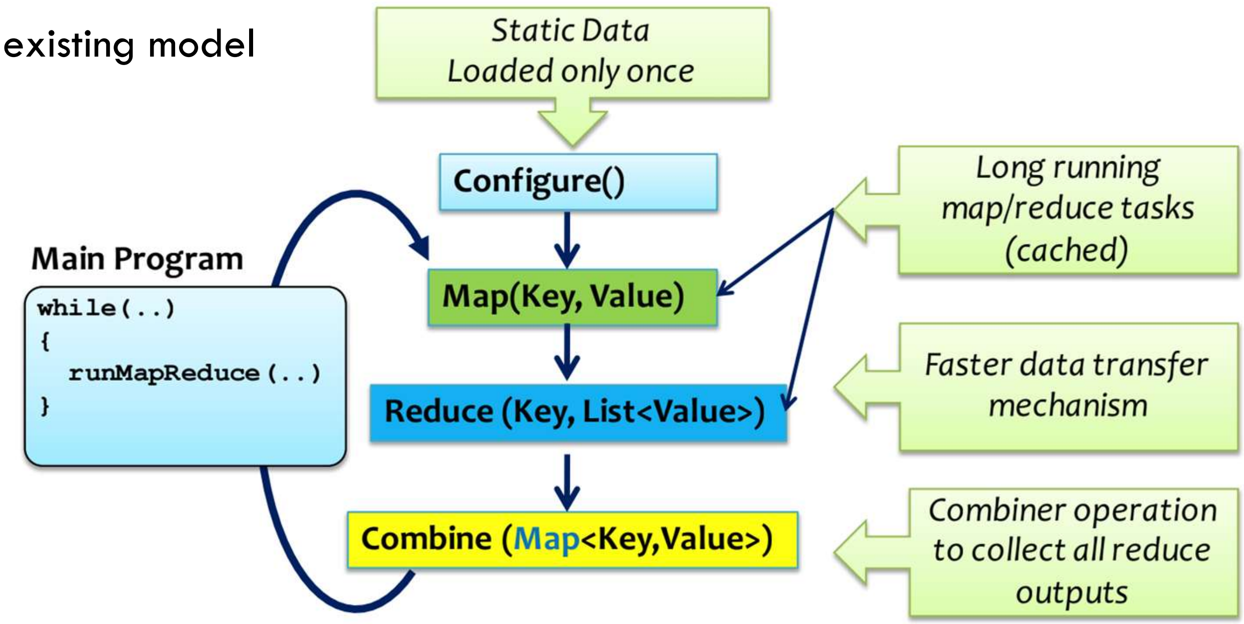
- Loop implemented on top of existing file-based single step map-reduce core
- Large overheads from
  - re-initialization of tasks
  - reloading of static data
  - communication and data transfers



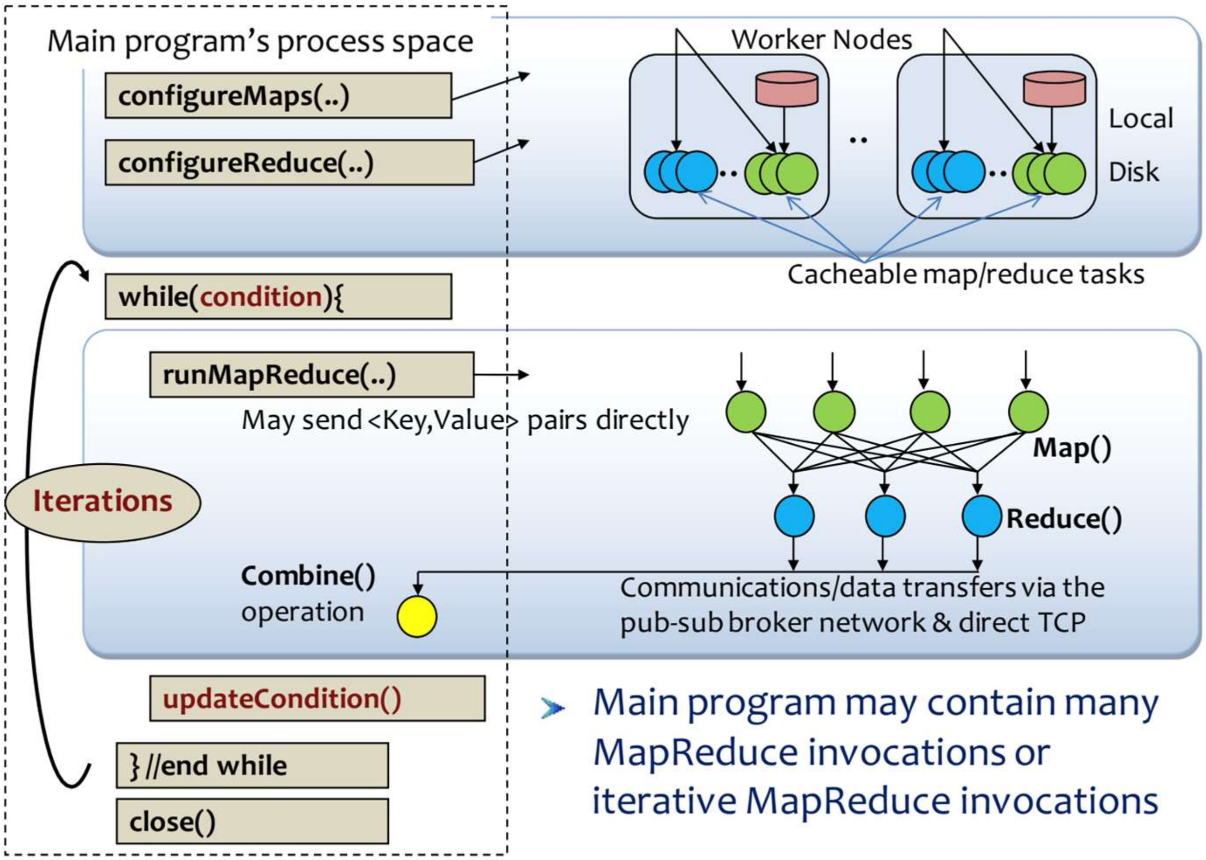
DistributedCache: https://hadoop.apache.org/docs/r2.6.3/api/org/apache/hadoop/filecache/DistributedCache.html

MapReduce++ : Iterative MapReduce

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



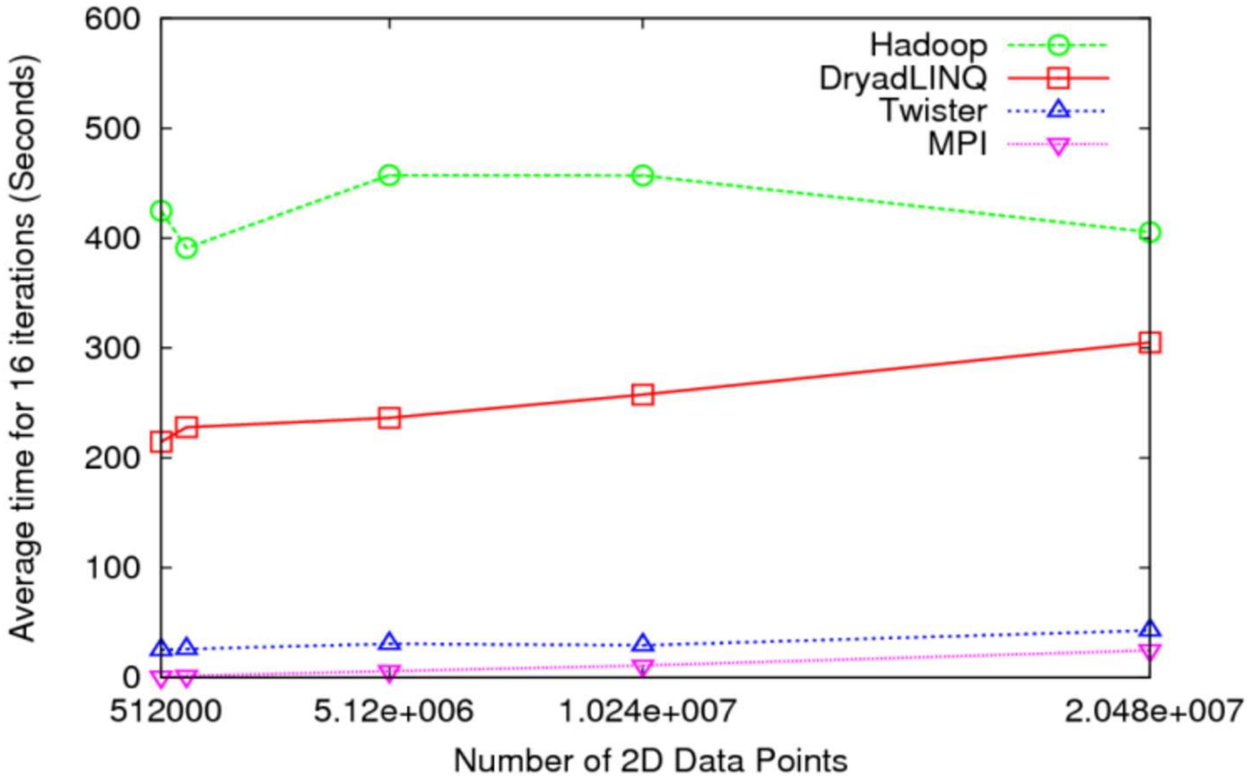
Example in Twister: Enables more APIs



- Main program may contain many MapReduce invocations or iterative MapReduce invocations

https://iterativemapreduce.weebly.com/ and ref Hwang Chap 6, Pg 351

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

Summary

- Different types of parallelism
- Data and tree parallelism —> map and reduce
- Basics of MapReduce programs with a historical sales data processing example
- Optimizations for iterative MapReduce requirements
- How does in-memory computing help for iterative MapReduce programming

Next Session:
Hadoop MapReduce and YARN