Logistics/Announcements

• Questions on PA3?
Today’s Lecture

1. MapReduce Implementation

2. Spark
1. MapReduce Implementation
Recall: The Map Reduce Abstraction for Distributed Algorithms

Map Reduce (Shuffle)

Distributed Data Storage

Map

Reduce
MapReduce: what happens in between?

- **Map**
  - Grab the relevant data from the source (parse into key, value)
  - Write it to an intermediate file

- **Partition**
  - Partitioning: identify which of $R$ reducers will handle which keys
  - Map partitions data to target it to one of $R$ Reduce workers based on a partitioning function (both $R$ and partitioning function user defined)

- **Shuffle & Sort**
  - Shuffle: Fetch the relevant partition of the output from all mappers
  - Sort by keys (different mappers may have sent data with the same key)

- **Reduce**
  - Input is the sorted output of mappers
  - Call the user `Reduce` function per key with the list of values for that key to aggregate the results
MapReduce: the complete picture
Step 1: Split input files into chunks (shards)

- Break up the input data into $M$ pieces (typically 64 MB)

Input files

Divided into $M$ shards
Step 2: Fork processes

- Start up many copies of the program on a cluster of machines
  - **One master**: scheduler & coordinator
  - Lots of workers

- Idle workers are assigned either:
  - map tasks (each works on a shard) – there are $M$ map tasks
  - reduce tasks (each works on intermediate files) – there are $R$
    - $R = \#$ partitions, defined by the user
Step 3: Run Map Tasks

- Reads contents of the input shard assigned to it
- Parses key/value pairs out of the input data
- Passes each pair to a user-defined map function
  - Produces intermediate key/value pairs
  - These are buffered in memory
Step 4: Create intermediate files

- Intermediate key/value pairs produced by the user’s map function buffered in memory and are periodically written to the local disk
  - Partitioned into $R$ regions by a partitioning function
Step 4a: Partitioning

- Map data will be processed by Reduce workers
  - User’s Reduce function will be called once per unique key generated by Map.

- We first need to sort all the (key, value) data by keys and decide which Reduce worker processes which keys
  - The Reduce worker will do the sorting

- **Partition function**
  *Decides which of R reduce workers will work on which key*
  - Default function: \( \text{hash(key)} \mod R \)
  - Map worker partitions the data by keys

- Each Reduce worker will later read their partition from every Map worker
Step 5: Reduce Task - sorting

- Reduce worker gets notified by the master about the location of intermediate files for its partition
- **Shuffle**: Uses RPCs to read the data from the local disks of the map workers
- **Sort**: When the reduce worker reads intermediate data for its partition
  - It sorts the data by the intermediate keys
  - All occurrences of the same key are grouped together
Step 6: Reduce Task - reduce

- The sort phase grouped data with a unique intermediate key

- User's **Reduce** function is given the key and the set of intermediate values for that key
  
  `<key, (value1, value2, value3, value4, ...)>`

- The output of the **Reduce** function is appended to an output file
Step 7: Return to user

- When all *map* and *reduce* tasks have completed, the master wakes up the user program.

- The *MapReduce* call in the user program returns and the program can resume execution.
  - Output of *MapReduce* is available in *R* output files.
MapReduce: the complete picture

We need a distributed file system!
2. Spark
Intro to Spark

- Spark is really a different implementation of the MapReduce programming model
- What makes Spark different is that it operates on Main Memory
- Spark: we write programs in terms of operations on resilient distributed datasets (RDDs).
- RDD (simple view): a collection of elements partitioned across the nodes of a cluster that can be operated on in parallel.
- RDD (complex view): RDD is an interface for data transformation, RDD refers to the data stored either in persisted store (HDFS) or in cache (memory, memory+disk, disk only) or in another RDD
RDDs in Spark

**RDD: Resilient Distributed Datasets**

- **Like a big list:**
  - Collections of objects spread across a cluster, stored in RAM or on Disk
- **Built through parallel transformations**
- **Automatically rebuilt on failure**

**Operations**

- **Transformations** (e.g. map, filter, groupBy)
- **Make sure input/output match**
MapReduce vs Spark

Map and reduce tasks operate on key-value pairs

Spark operates on RDD
RDDs

- Partitions are recomputed on failure or cache eviction
- Metadata stored for interface:
  - Partitions – set of data splits associated with this RDD
  - Dependencies – list of parent RDDs involved in computation
  - Compute – function to compute partition of the RDD given the parent partitions from the Dependencies
  - Preferred Locations – where is the best place to put computations on this partition (data locality)
  - Partitioner – how the data is split into partitions
Lazy computations model

Transformation cause only metadata change

RDDs
• Directed Acyclic Graph – sequence of computations performed on data
• Node – RDD partition
• Edge – transformation on top of the data
• Acyclic – graph cannot return to the older partition
• Directed – transformation is an action that transitions data partitions state (from A to B)
Example: Word Count

```scala
> lines = sc.textFile("hamlet.txt")
> counts = lines.flatMap(lambda line: line.split(" "))
  .map(lambda word: (word, 1))
  .reduceByKey(lambda x, y: x + y)
```
Spark Architecture
Spark Driver

- Entry point of the Spark Shell (Scala, Python, R)
- The place where SparkContext is created
- Translates RDD into the execution graph
- Splits graph into stages
- Schedules tasks and controls their execution
- Stores metadata about all the RDDs and their partitions
- Brings up Spark WebUI with job information
Spark Executor

- Stores the data in cache in JVM heap or on HDDs
- Reads data from external sources
- Writes data to external sources
- Performs all the data processing
Dag Scheduler

- General task graphs
- Automatically pipelines functions
- Data locality aware
- Partitioning aware to avoid shuffles

![Diagram](image)

- A: Stage 1
- B: groupBy
- C: map
- D: filter
- E: join
- F: Stage 3

= RDD  = cached partition
More RDD Operations

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin
- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip
- sample
- take
- first
- partitionBy
- mapWith
- pipe
- save
...
Spark’s secret is really the RDD abstraction

**RDD: Resilient Distributed Datasets**

- **Like a big list:**
  - Collections of objects spread across a cluster, stored in RAM or on Disk
- **Built through parallel transformations**
- **Automatically rebuilt on failure**

**Operations**

- Transformations (e.g. map, filter, groupBy)
- Make sure input/output match