Computational Framework: MapReduce

Big data challenges

- Computational complexity
  - Any processing requiring a **superlinear number of operations** may easily turn out **unfeasible** (*e.g. O(n^3) with big datas*)
  - If input size is really huge, just **touching all data items** is already time consuming
  - For computation-intensive algorithms, exact solutions may be too costly. **Accurancy-efficiency tradeoffs** (*give up accurancy for better efficiency*)
- Effective use of parallel/distributed platforms:
  - Sepcialized high-performance architectures are costly and become rapidly obsolete
  - Fault-tollerance becomes serius issue: **low Mean-Time Between Failures** (*MTBF*)
  - Parallel/distributed programming requires **high skills**

MapReduce

- Introduced by Google 2004
- Programming framework for handling big data
- Employed in **many application scenarios** on **clusters of commodity processors** and **cloud infrastructures**
- Main features:
  - Data centric view
  - Inspired by functional programming (*map, reduce functions*)
  - Ease of programming. Messy details are hidden to the programmer
- Main implementation: **Apache Hadoop**
  - extremely slow
  - High inefficiency
- Hadoop ecosystem: several variant and extensions aimed at improving Hadoop's performance (*e.g. Apache Spark*)

Typical cluster architecture

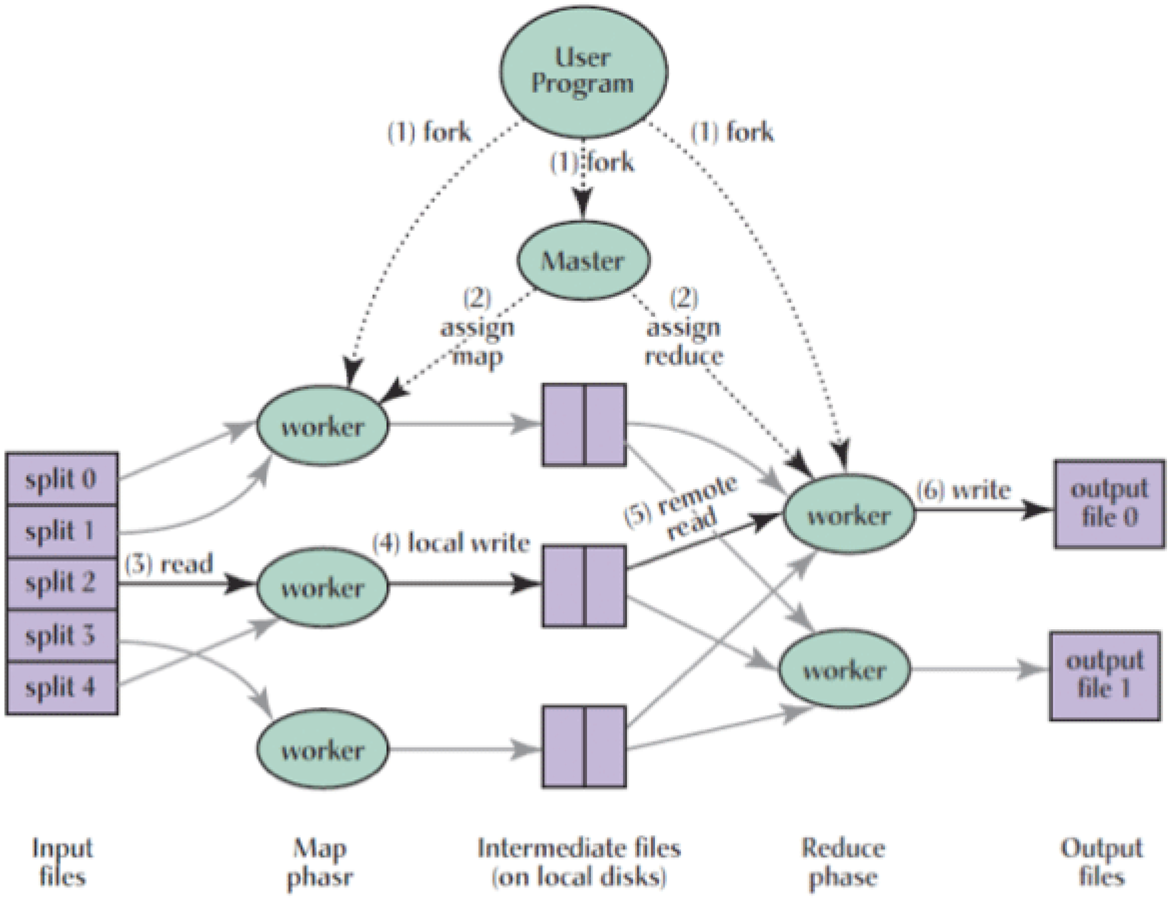
- Racks of 16-64 **compute-nodes** connected by fast switches
- Distributed File System
  - Files divided into chunks
  - Each chunk repicated with replicas in different nodes and, possibly, in different racks
  - The distribution of the chunks of a file is represented into a master node file wichi is also replicated. A directory records where all master nodes are

MapReduce computation

- Computation viewed as a **sequence of rounds**.
- Each round transforms a set of **key-value pairs** into another set of **key-value pairs** (*data centric view!*) through the following phases

- Map phase: a user specified **map-function** is applied separately to each input key-value pair and procudes other key-value pairs referred as **intermediate key-value pairs**
- Reduce phase: the **intermediate key-value pairs** are **grouped by key** and a user-specified **reduce function** is applied separately to each group of key-value pairs with the same key, producing other key-value pairs which is the output of the round

MapReduce Round



- Starts and ends in HDFS (_Hadoop Distributed File System)
- Data are split in chunks
- Workers are actual machines
- local disks for intermediate files are the memory of the workers

Dealing with faults

- The Distributed File System is fault-tolerant
- Master pings workers periodically to detect failures
- Worker failure:
    - Map tasks completed or in-progress at failed worker are reset to idle and will be rescheduled. Note that even if a map task is completed, the failure of the worker makes its output unavailable to reduce tasks, hence it must be rescheduled.
    - Reduce tasks in-progress at failed worker are reset to idle and will be rescheduled.
- Master failure: the whole MapReduce task is aborted

Specification of a MapReduce (MR) algorithm

1. Specify what the input and the output are
2. Make clear the sequence of rounds
3. For each round
   - input intermediate and output key-value pairs need to be clear
   - functions in the map and reduce phases need to be clear
4. Enable analysis

Key performance indicators

1. Number of rounds
2. Local space M_L: maximum amount of space required by any map or reduce function executed by the algorithm for storing input and temporary data (*but not the output since it will be stored in the local disk or in the HDFS depending on the step of the mapReduce*)
3. Aggregate space M_A: maximum amount of space which, at any time during the execution of the algorithm, is needed to store all data required at that time or at future times

Observation

- The indicators are usually estimated through asymptotic analysis as functions of the instance parameters (*e.g. imput size*). The analysis could be worst-case (*algorithmic-like*) or probabilistic.

- In general, the number of rounds R depends on the instance, on M_L, and on MA. Typically, the larger M_L and M_A, the smaller R.

Design goals for MapReduce algorithms

Theorem:

For every computational problem solvable sequentially with space complexity **S(|input|)** there exists a 1-round MapReduce algorithm with M_L = M_A = Θ (S(|input|))

- For efficiency, the design of an MR algorithm should aim at:
  - few rounds
  - Sublinear local space
  - Linear aggregate space, or only slightly superlinear
  - Polynomial complexity of each map or reduce function

Basic primitives and tecniques

Word Count

1. **Input**: collection of text documents containing N words overall. Each document is a key-value pair, whose key is the document's nae and the value is its content.
2. **Output**: The set of pairs **(w,c(w))** where **w** is a word occurring in the documents, and **c(w)** is the number of occurrences of **w** in the documents.

Round 1:

- Map phase: for each document produce the set of intermediate pairs **(w,1)**, one for each occurence of a word **w** N.B.: the wordk is the key of the pair.
- Reduce phase: for each word **w**, gather all intermediate pairs **(w,1)** and produce the pair **(w,c(w))** by summing all values of the pair.

worst-case analysis

respect to the input size N

- R = 1
- M_L = O(N). Bad case: only one word occurs repeated N times over all documents
- M_A = O(N).

Observation: The algorithm does not satisfy the aforementioned design goals: in particular, it does not attain sublinear local space

Improved word count

For each document the map function produces ine pair for each word with the number of occurrences of said word in the document.

Let N_i be the number ov words in D_i. The optimization yields:

- R = 1
- M_L = O(max_{i=1,k} N_i+k)
- M_A = O(N)

Observation:

- The sublinear local space requirement is satisfied as long as N_i = o(N) for each i, and k = o(N)
- by treating each document as an individual key-value pair we have that for any algorithm M_L= Ω (max_{i=1,k}N_i). Implies that only the O(k) additive term can be removed

Partitioning Technique:

When some aggregation functions may potentially receive large inputs (*e.g. large k*) or skewed ones it is advisable to partition the input, either deterministically or randomly, and perform aggregation in stages.

This can be done with:

- An improved version of word count
- A general category counting primitive

Improved Word count 2

Idea: partition intermediate pairs in o(N) groups and compute counts in two stages

Round 1:

- **Map phase**: for each document *D_i*, produce the intermediate pairs (x,(w,c_i(w))), one for every word *w in D_i*, where *x* is a random value in [0,sqrt(*N*)) and c_i(w) is the number of occurrences of *w* in *D_i*

- **Reduce phase**: For each key *x* gather all pairs (x,(w,c_i(w))), and for each word *w* occurring in these pairs produce the pair (w,c(x,w)) where c(x,w) = sum c_i(w). Now, *w* is the key for (w,c(x,w))

Round 2:

- **Map phase**: identity function
- **Reduce phase**: for each word *w*, gather the at most sqrt(*N*) pairs (w,c(x,w)) resulting at the end of the previous round, and produce the pair (w,sum c(x,w))

Analysis

Let *m_x* be the numberof intermediate pairs with key *x* produced by the Map phase of Round 1, and let m = max_x m_x. As before, let *N_i* be the number of words in *D_i*. We have

- R = 2
- M_L = O(max_{i-1,k} *N_i* + m + sqrt(*N*))
- M_A = O(*N*)

Estimate of *m* for Word Count 2

Theorem:

```
Suppose that the key assigned to the intermediate pairs in Round 1 are i.i.d
random variables with uniform distribution in [0,sqrt(_N_)). Then, with
probability at least 1 - 1/_N_^5

m = O(sqrt(_N_))
```

Therefore, from the theorem and the preceding analysis whe get

$$M_L = O\left(\max_{i=1,k} N_i + \sqrt{N}\log N\right),$$

Category counting

Suppose that we are given a set *S* of *N* objects, each labeled with a category from a given domain, and we want to count how many objects belong to each category.

Round 1:

- *Map phase*: map each pair $(i, (o_i, \gamma_i))$ into the intermediate pair $(i \bmod \sqrt{N}, (o_i, \gamma_i))$ (mod = reminder of integer division)
- *Reduce phase*: For each key $j \in [0, \sqrt{N})$ gather the set (say $S^{(j)}$) of all intermediate pairs with key j and, for each category γ labeling some object in $S^{(j)}$, produce the pair $(\gamma, c_j(\gamma))$, where $c_j(\gamma)$ is the number of objects of $S^{(j)}$ labeled with γ.

Round 2:

- *Map phase*: identity function
- *Reduce phase*: for each category γ, gather the at most $\sqrt{N}$ pairs $(\gamma, c_j(\gamma))$ resulting at the end of the previous round, and produce the pair $(\gamma, \sum_j c_j(\gamma))$.

Trading accurancy for efficiency

There are problems for which **exact MR algorithm** may be **too costly**, namely they may require a large number of rounds, or large local space, or large aggregate space. These algorithms become impractical for very large inputs.

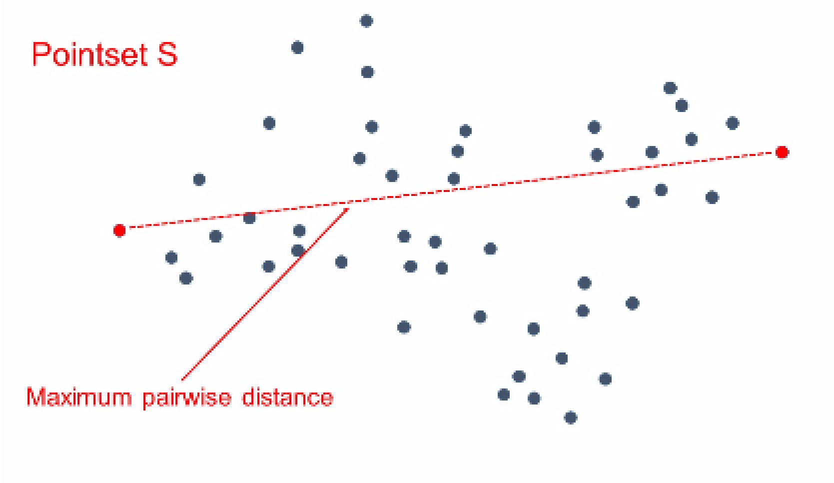
In these scenarios, **giving up exact solutions may greately improve efficiency**.

Maximum pairwise distance

Suppose that we are given a set *S* of *N* points from some metric space and we want to determine the maximum distance between two points, for a given distance function d(.,.)

Input: Set *S* of *N* points represented by pairs *(i, x_i)*, for 0<=i<=*N*, where x_i is the i-th point

Output: d_max = max d(x_i,x_j)



We can substantially reduce the aggregate space requirements if we tolerate a factor 2 error in the estimate of d_max. For an arbitrary point x_i define

$$d_{\max}(i) = max_{0 \le j < N} d(x_i, x_j).$$

Lemma

For any $0 \le i < N$ *we have* $d_{\max} \in [d_{\max}(i), 2d_{\max}(i)]$.

Round 1:

- *Map phase*: map each pair (i, x_i) into the intermediate pair $(i \bmod \sqrt{N}, (i, x_i))$. Also, create the $\sqrt{N} - 1$ pairs $(j, (0, x_0))$, for $1 \le j < \sqrt{N}$.
- *Reduce phase*: For each key $j \in [0, \sqrt{N})$ gather the set $S^{(j)}$ of all intermediate pairs with key j (which include $(j, (0, x_0))$) and produce the pair $(0, d_{\max}(0, j))$ where $d_{\max}(0, j)$ is the maximum distance between x_0 and the points associated with pairs in $S^{(j)}$

Round 2:

- *Map phase*: identity.
- *Reduce phase*: gather all pairs resulting at the end of the previous round, and output $d_{\max}(0) = \sum_{0 \le j < \sqrt{N}} d_{\max}(0, j)$.

Analysis: $R = 2$, $M_L = O\left(\sqrt{N}\right)$, $M_A = O(N)$.

Trading rounds for space efficiency

Minimizing the number of rounds may sometimes force large space requirements. In these cases, one could aim at devising algorithms which feature suitable tradeoffs between local and/or aggregate space requirements and number of rounds.

We already saw an example of this kind of tradeoffs.

Exploiting samples

The first question that should be asked when facing a big-data processing task is the following

Can small subsets of the data help speeding up the task?

In many cases, the answer is YES! Specifically, a small sample, suitably extracted, could be exploited for the following purposes

- **To subdivide the dataset in smaller subsets** to be analyzed separately
- To provide a **succint yet accurate representation of the whole dataset**, which contains a good solution to the problem and filters out noise and outliers, thus allowing the execution of the task on the sample

Sorting

Input: Set $S = \{s_i \ : \ 0 \le i < N\}$ of N distinct sortable objects (each s_i represented as a pair (i, s_i))

Output: Sorted set $\{(i, s_{\pi(i)}) \ : \ 0 \le i < N\}$, where π is a permutation such that $s_{\pi(1)} \le s_{\pi(2)} \le \cdots \le s_{\pi(N)}$.

MR Samplesort algorithm. The algorithm is based on the following idea:

- Fix a suitable integral design parameter *K*
- Randomly select some objects (*K* on average) as splitters
- Partition the object into subsets based on the ordered sequence of splitters
- Sort each subset separately and compute the final ranks

MR Samplesort

Round 1:

- **Map phase**: for each pair (i,s_i) do the following: create the intermediate pair (i mod *K*, (0,s_i)) and, with probability p=*K*/*N* independently of other objects, select s_i as a splitter. If s_i is selected as splitter then create *K* additional intermediate pairs (j,(1,s_i)), with 0<=j<*K*. Suppose that *t* objects are selected as splitters
- **Reduce phase**: for 0<=j<*K* do the following: gather all intermediate and splitter pairs with key j; sort the splitters; for every o<=*l*<=*t* and every intermediate pair (j,(0,s)), with x_l< s< x_{l+1}, produce the pair (l,s) with key *l*

Round 2:

- **Map phase**: identity
- **Reduce phase**: for every 0< *l* < *t* gather, from the output of the previous round, the set of pairs S^*l* = {(*l*,s)}, compute N_ *l*, and create t+1 replicas of N_ *l*

Round 3:

- **Map phase**: identity
- **Reduce phase**: for every 0<= *l* <= *t* do the following: gather S^*l* and the values *N* _0, *N* _1, ..., *N* _t; sort S^*l*; and compute the final output pairs for the objects in S^*l* whose ranks start from 1+{sum from 0 to *l*-1}*N* _h

Analysis of MR Samplesort

- Number of rounds: $R = 3$
- Local Space M_L:
  - Round 1: $O(t + N/K)$, since each reducer must store all splitter pairs and a subset of N/K intermediate pairs.
  - Round 2: $O(\max\{N_\ell \ ; \ 0 \le \ell \le t\})$ since each reducer must gather one $S^{(\ell)}$.
  - Round 3: $O(t + \max\{N_\ell \ ; \ 0 \le \ell \le t\})$, since each reducer must store all N_ℓ's and one $S^{(\ell)}$.

  $\Rightarrow$ overall $M_L = O(t + N/K + \max\{N_\ell \, ; \, 0 \le \ell \le t\})$
- Aggregate Space M_A: $O(N + t \cdot K + t^2))$, since in Round 1 each splitter is replicated K times, and in Round 3 each N_ℓ is replicated $t + 1$ times. The objects are never replicated.

Lemma

With reference to the MR SampleSort algorithm, for any $K \in (2 \ln N, N)$ the following two inequalities hold with high probability (i.e., probability at least $1 - 1/N$):

1. $t \leq 6K$, *and*
2. $\max\{N_i \; ; \; 0 \leq \ell \leq t\} \leq 4(N/K) \ln N$.

Theorem

By setting $K = \sqrt{N}$, MR SampleSort runs in 3 rounds, and, with high probability, it requires local space $M_L = O\left(\sqrt{N} \ln N\right)$ and aggregate space $M_A = O(N)$.