MapReduce

A programming model for processing large datasets

Prof. Tahar Kechadi

School of Computer Science & Informatics

Outline

- **MapReduce**, the model
- **Hadoop**, an open-source implementation

- **Pig**, a high-level abstraction layer

What is MapReduce?

- A data-parallel programming model designed for high scalability and resiliency
- Pioneered by Google. Also designed Percolator – *incrementally processing updates*
  - *Implementation in C++*
- Popularised by the open-source Hadoop project
  - Used at Yahoo!, Facebook, Amazon, …

Used / has been used for

- At Google
  - Index construction for Google Search, Article clustering for Google News, Statistical machine translation
- At Yahoo!
  - “Web map” powering Yahoo! Search, Spam detection for Yahoo! Mail
- At Facebook
  - Ad optimisation, Spam detection
- In research
  - Astronomical image analysis, Bioinformatics, Analysing Wikipedia conflicts, Natural language processing, Particle physics, Ocean climate simulation
- Among others…

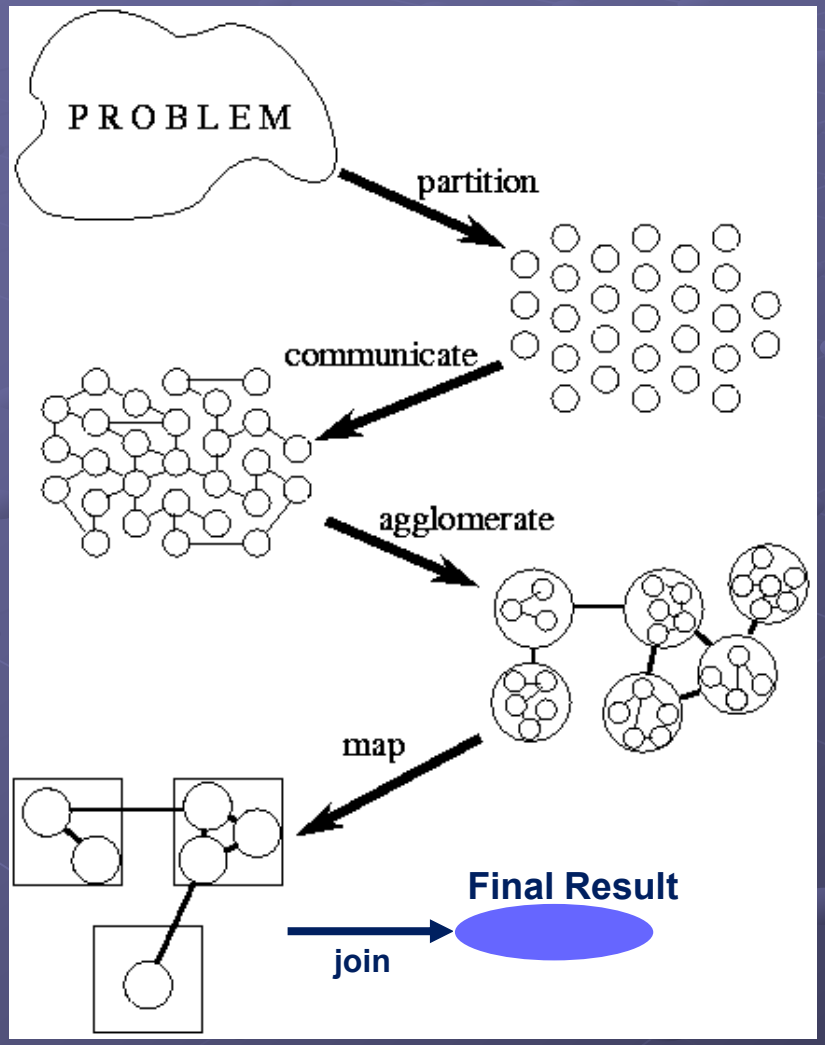
Parallel Programming

- Process of developing programs that express what computations should be executed in parallel.
- Parallel computing
  - use of two or more processors (computers), ***usually within a single system***, working simultaneously to solve a single problem
- Distributed computing
  - any computing that involves ***multiple computers remote from each other*** that each have a role in a computation problem or information processing

Parallel vs. Distributed Computing

Characteristic	*Parallel*	*Distributed*
Overall Goal	Speed	Convenience
Interactions	Frequent	Infrequent
Granularity	Fine	Coarse
Reliable	Assumed	Not Assumed

Parallel Programming Methodical Design



- Partitioning
  - Computation, data are decomposed into small tasks.
- Communication
  - Coordinate task execution
- Agglomeration
  - Tasks are combined to larger tasks
- Map
  - Assigned to a processor
- Join

Parallel Programming Paradigms

Data parallelism

- All tasks apply the same set of operations to different data
- Example:

```
for i ← 0 to 99 do
    a[i] ← b[i] + c[i]
endfor
```

- Operations may be performed either synchronously or asynchronously
- Fine grained parallelism
- *Grain Size* is the average number of computations performed between communication or synchronization steps

Parallel Programming Paradigms

Task parallelism

- Independent tasks apply different operations to different data elements
- Example

$$a \leftarrow 2$$
$$b \leftarrow 3$$
$$m \leftarrow (a + b) / 2$$
$$s \leftarrow (a^2 + b^2) / 2$$
$$v \leftarrow s - m^2$$

- Concurrent execution of tasks, not statements
- Problem is divided into different tasks
- Tasks are divided between the processors
- Coarse grained parallelism

Parallel Programming Examples

- Adding a sequence of n numbers
  - Sum = A[0] + A[1] + A[2] + … + A[n-1]
  - Code:

```
sum ← 0
for i = 0 to  n-1 do
      sum ← sum + A[i]
end for
```

  - Sequential or parallel?

Parallel Programming Example I

- Adding a sequence of n numbers
  - Sum = A[0] + A[1] + A[2] + … + A[n-1]
  - Code:

```
sum ← 0
for i = 0 to  n-1 do
      sum ← sum + A[i]
end for
```

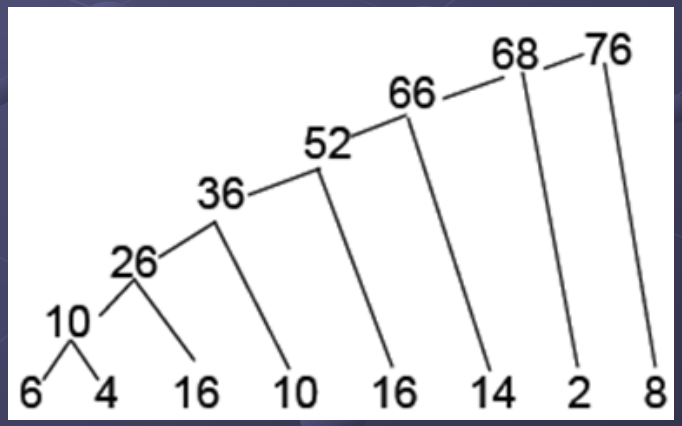
  - Example: : 6 4 16 10 16 14 2 8

    sum ← 0 + 6

    sum ← 6 + 4

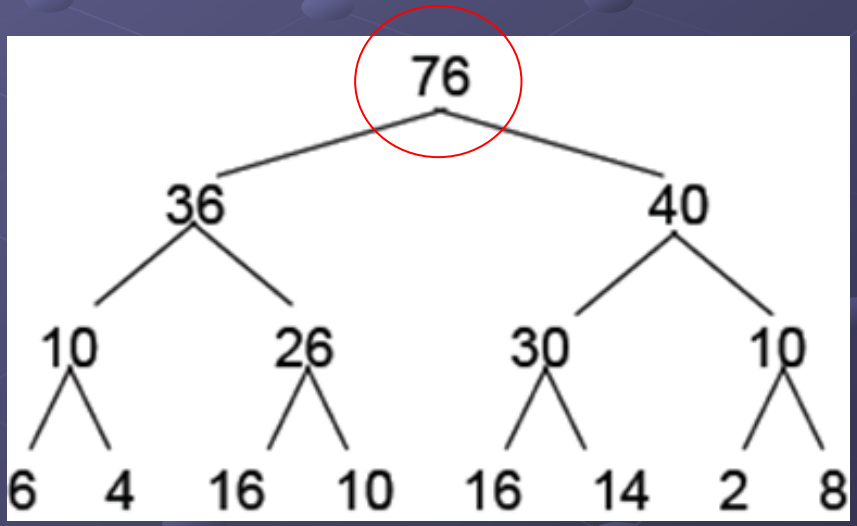
    sum ← 10 + 4

    …



Parallel Programming Example I

- Adding a sequence of n numbers
  - Sum = A[0] + A[1] + A[2] + … + A[n-1]
  - In parallel:
    - Add pairs of values producing 1st level results,
    - Add pairs of 1st level results producing 2nd level results
    - Sum pairs of 2nd level results
    - …



Parallel Programming Example II

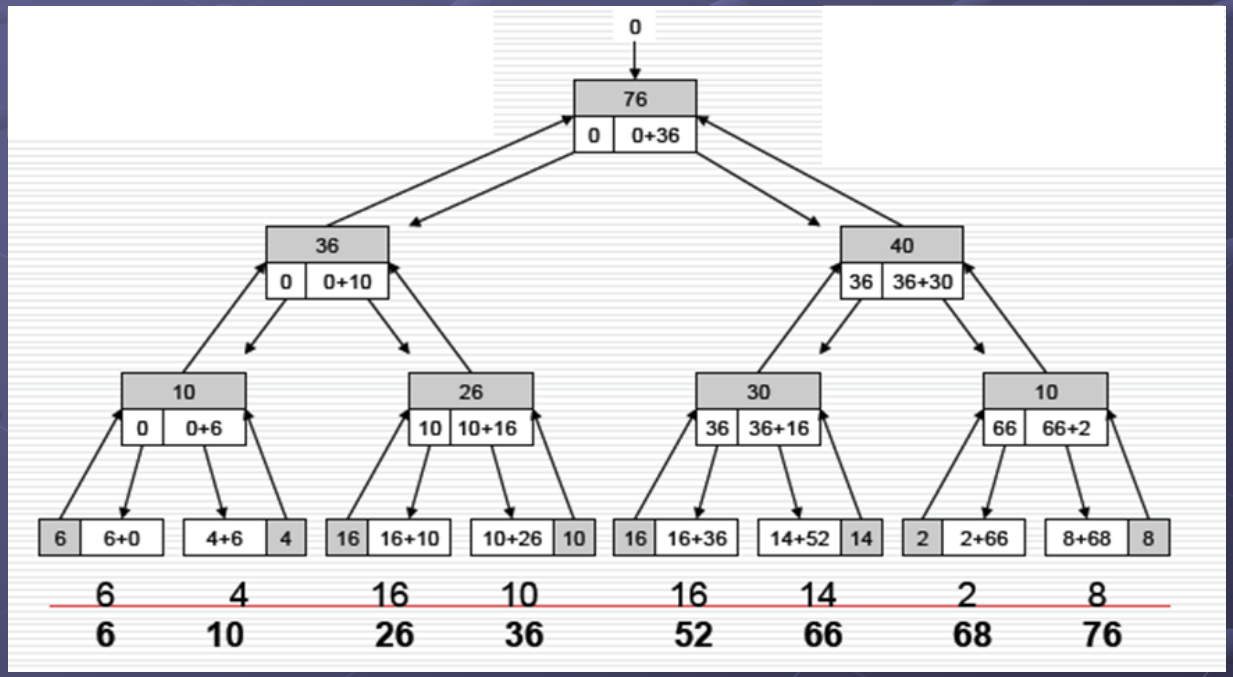
- Computing the prefix sums
  - A[i] is the sum of the first i+1 elements
  - Code:

```
for i = 1 to  n-1 do
    A[i] ← A[i] + A[i-1]
end for
```

  - So:
    - A[0] is unchanged
    - A[1] = A[1] + A[0]
    - A[2] = A[2] + (A[1] + A[0])
    - …
    - A[n-1]= A[n-1] + (A[n-2] + (…(A[1] + A[0]) …)
  - Example:
    - Input: 6 4 16 10 16 14 2 8
    - Output: 6 10 26 36 52 66 68 76

Parallel Programming Example II

- Computing the prefix sums
  - A[i] is the sum of the first i+1 elements
  - Example:
    - Input: 6 4 16 10 16 14 2 8
    - Output: 6 10 26 36 52 66 68 76



Traditional HPC systems

- CPU-intensive computations
  - Relatively small amount of data
  - Tightly-coupled applications
  - Highly concurrent I/O requirements
  - Complex message passing paradigms such as MPI, PVM…
  - Developers might need to spend some time designing for failure

Challenges

- Data and storage
  - Locality, computation close to the data
- In large-scale systems, nodes fail
  - MTBF (Mean time between failures) for 1 node = 3 years
  - MTBF for 1000 nodes = 1 day
  - Solution: Built in fault-tolerance
- Commodity network = low bandwidth
- Distributed programming is hard
  - Solution: simple data-parallel programming model: users structure the application in “map” & “reduce” functions, system distributes data/work and handles faults
  - Not all applications can be parallelised: tightly-coupled computations

What requirements?

- A simple data-parallel programming model, designed for high scalability and resiliency
  - Scalability to large-scale data volumes
  - Automated fault-tolerance at application level rather than relying on high-availability hardware
  - Simplified I/O and tasks monitoring
  - All based on cost-efficient commodity machines (cheap, but unreliable), and commodity network

Core concepts

- Data spread in advance, persistent (in terms of locality), and replicated
- No inter-dependencies / shared nothing architecture
- Applications written in two pieces of code
  - And developers do not have to worry about the underlying issues in networking, jobs interdependencies, scheduling, etc…

The model

- A map function processes a key/value pair to generate a set of intermediate key/value pairs
  - Divides the problem into smaller ‘intermediate key/value’ pairs
- The reduce function merge all intermediate values associated with the same intermediate key
- Run-time system takes care of:
  - Partitioning the input data across nodes (blocks/chunks typically of 64Mb to 128Mb)
  - Scheduling the data and execution
  - Node failures, replication, re-submissions
  - Coordination among nodes