Introduction to Parallel and Distributed Processing

Introduction to Higher-level Programming Models

Jaroslaw 'Jaric' Zola

http://www.jzola.org/

Suggested Reading

- Dean, J. and Ghemawat, S.
  "MapReduce: Simplified Data Processing on Large Clusters"
  http://research.google.com/archive/mapreduce.html

Live Is Too Hard

- MPI and similar models are extremely well suited for HPC
- But they fail for data-driven problems
  - Complex low-level API
  - No fault tolerance
  - No elasticity
  - No “easy” I/O
- With the emergence of big data, these attributes became really important:
  - Large unstructured data
  - Large networks
  - Data streams
  - Etc.

Hardware Fails

- Jeff Dean (Google):

  "In each cluster's first year, it's typical that 1,000 individual machine failures will occur; thousands of hard drive failures will occur; one power distribution unit will fail, bringing down 500 to 1,000 machines for about 6 hours; 20 racks will fail, each time causing 40 to 80 machines to vanish from the network; 5 racks will "go wonky," with half their network packets missing in action; and the cluster will have to be rewired once, affecting 5 percent of the machines at any given moment over a 2-day span, Dean said. And there's about a 50 percent chance that the cluster will overheat, taking down most of the servers in less than 5 minutes and taking 1 to 2 days to recover."

Google Map/Reduce

- In 2004 Google published their answer to the problem
- Map/Reduce programming model
  plus supporting middleware:
  - Use stateless (shared nothing) approach
  - Express algorithms using mappers and reducers
  - Provide distributed I/O, fault tolerance, scheduling and communication in the executing environment
- Easy to use for embarrassing parallelism

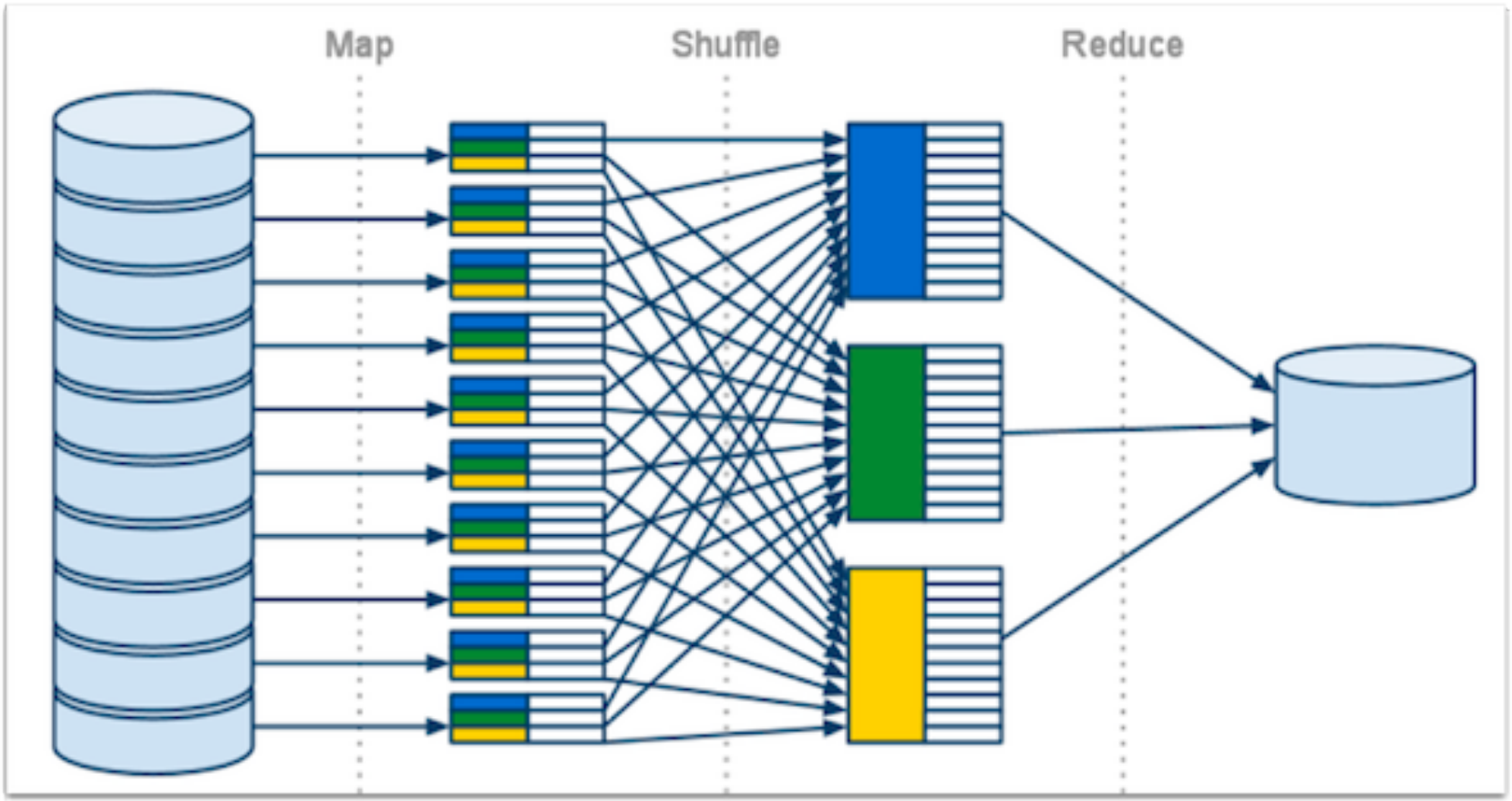
Map/Reduce Model

- Input/output represented via $(key, value)$ pair
- Algorithms expressed using two functions, `map` and `reduce`:
  $map : K \times V \rightarrow K' \times V'$
  $reduce : K' \times [V' \times \ldots \times V'] \rightarrow K'' \times V''$

Map/Reduce Implementation

- Programming model not new (Lisp, MPI, etc.)
- But new approach at the system level:
  - Stateless approach, hence nodes can fail and recover easily, platform can be scaled up/down
  - Distributed and not parallel file system
  - Ideal for commodity clusters

Map/Reduce Model



Map/Reduce Example

- We are given a list of edges E of a directed graph, we want to count in- and out-degree of each node:

```
1  map(K k, V v):
2    emit ((v, "in"), 1)
3    emit ((k, "out"), 1)
4
5  reduce(K k, Iter i)
6    S = 0
7    for i in Iter:
8      S += 1
9    emit (k, S)
```

For Fun

- Consider the problem of connected components over the list of edges. How would you express initialization step in Map/Reduce?