
# MapReduce

Jens Andersson

# Agenda

- Motivation and Objectives
- MapReduce Programming model
  - Example
- The Google Way
- Apache Haadop
  - Pig
  - Hive
  - Spark

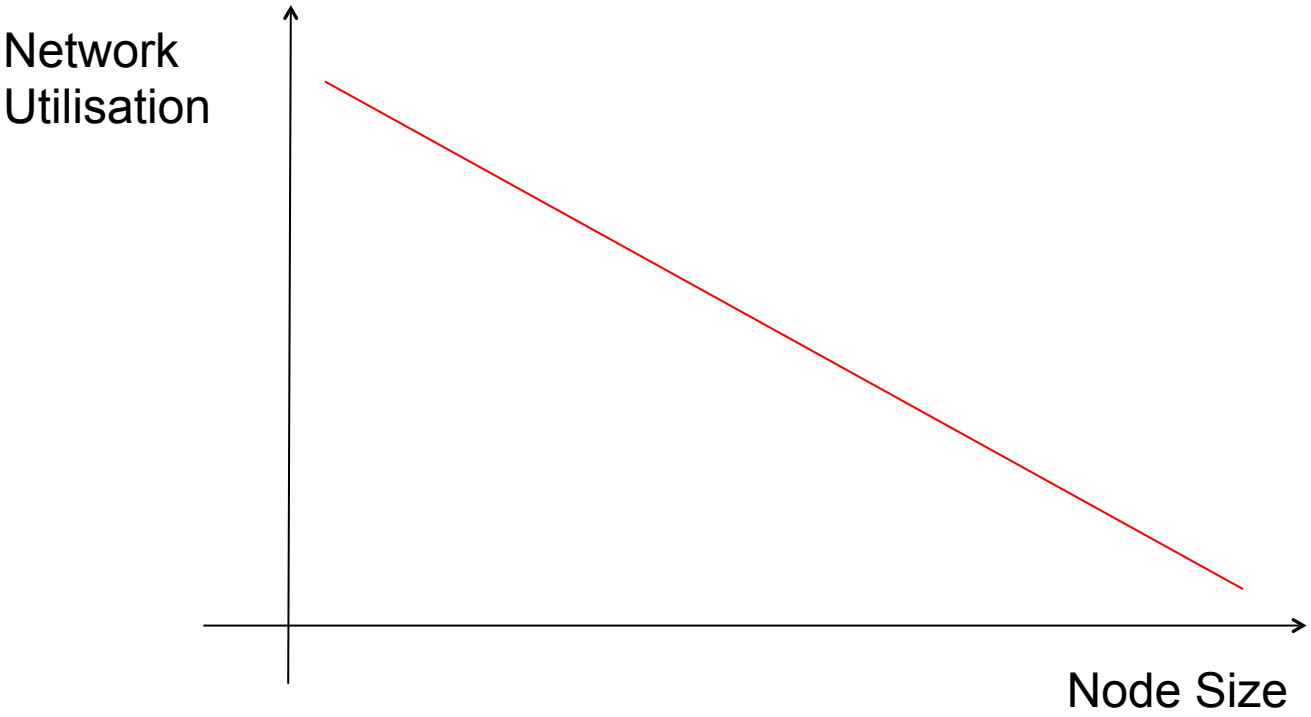
# Motivation and Objectives

- **Parallel processing** of **huge data sets**
- How to distribute data to processes/processors
  - Load balancing
  - Parallel execution / Reduce execution time
  - Fault tolerance

# MapReduce: Programming model

- Two-stage, mostly disk-based, paradigm
- Map function:
  - Process input key/value pairs to create intermediate key/value pairs
  - $(k_n,v_n)$ -> list$(k_i,v_i)$
- Reduce function:
  - Process input intermediate keys and all values for that key into one (few) values for that key
  - $(k_i,\text{list}(v_i))$ -> list$(v_i)$
- If clusters:
  - Add “Shuffle”/Distribute Map and Reduce functions

# Node Size vs Network Utilisation



# Example

- Usage data from Video on Demand (VoD) network
  - 1 month of data
  - 17,000,000 request
  - 560,000 active accounts
  - 20,000 requested programs
  - 80 different channels
  - 2 GB of data in one text file
  - For each request
    - Date and time, Account, Program meta data (name, season, episode, length .. …), Channel
- *Disclaimer:*
  - *fully possible to run on a normal PC*
  - *basic data reduction approx 2-3 hours of execution time*

# Example (cont …)

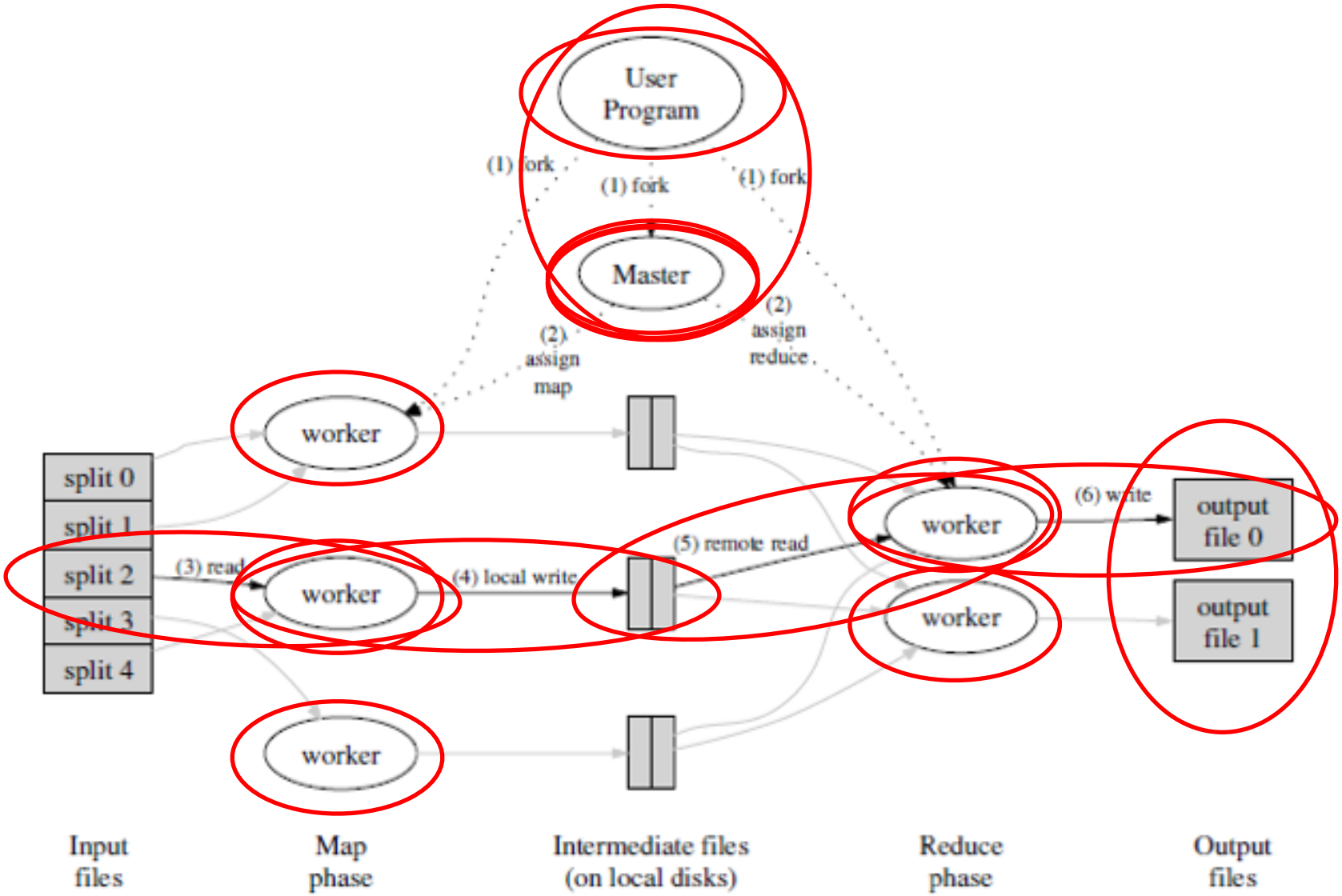
- Longitudinal study of number of request per active account
- **Map** all requests for one account as one set
  - Key = account id
  - Data = request meta data
- **Distribute** (“Shuffle”) sets **per key** to cluster nodes
- **Reduce** data per unique accounts
  - Example: Calculate duration of sessions of consecutive requests
- **Present** reduced data

# Example (cont …)

- Our example:
  - Map:
    $(req_n,(account,start,stop)_n)$ -> list($account_i$, $(req,start,stop)_i$)
  - Reduce:
    $(account_i$, list$((req,start,stop)_i))$ ->
    list$(account,avg(session))_i$

# The Google Way

- J. Dean and S. Ghemawat, ”MapReduce: Simplified Data Processing on Large Clusters”
- Adapted to Google’s commodity cluster node and Googles File System (GFS)
- Locality
  - Keep track of physical location of stored data to minimise network utilisation
- Task granularity
  - How to distribute Map and Reduce job to the cluster
  - Optimisation, f(node capacity, physical location)
  - Goal: Load balancing, failure recovery


User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
split 0
split 1
split 2
split 3
split 4
(3) read
worker
(4) local write
worker
(5) remote read
worker
worker
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

# What if …?

- Worker failure (master recognises no response from worker)
  - Reset to idle state
  - Reschedule lost task on available worker
- Master failure
  - Current master periodically writes checkpoints
  - On failure
    - Alt 1: Assign new master and restart at last checkpoint
      Any worker that detects dead master can be new master
    - Alt 2: Since failure of master is unlikely, kill job and start over.

# Locality

- GFS stores several replicas (3) of each file 64 MB block on different machines
- Master schedules Map task on workers that already carry a replica.

# Backup Tasks

- Not all workers perform on top
  - ”Stragglers”
- When close to completion
  - Master schedules backup execution of remaining *in-progress* task on idle worker
  - Task is marked completed whenever original or backup has completed the task.

# Apache Hadoop

- From http://hadoop.apache.org/
  ”The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.”
- Relies on two-stage disk-based MapReduce Paradigm (compare with Google)
- SW lib for distributed processing of large datasets
  - Hadoop Common
    - ”root”
  - Hadoop Distributed File System (HDFS™)
  - Hadoop YARN
    - Job scheduling, cluster resource management for MapReduce
  - Hadoop MapReduce
    - YARN-based system for parallel processing

# Pig, Hive, Spark

**Apache Pig**

- High-level platform on top of Haadop
- Pig Latin:
  - Language for the platform, similar to SQL
  - Procedural, fits well into pipe-line paradigm
- Unlike SQL Pig can split a data processing stream and apply different operators to each split.

- Developed at Yahoo but moved to Apache Software Foundation

# Pig, Hive, Spark

**Apache Hive**

- Report and analysis infrastructure built on top of Haadop
- Data summarisation, query, analysis
- SQL-like language: HiveQL
- Offers basic support for indexes

- Developed by Facebook
- Included in Amazon Elastic MapReduce

# Pig, Hive, Spark

**Apache Spark**

- Open-source cluster computing framework
- Works in memory (contrary to two-stage disk-based paradigm), thus much faster.
- Requires cluster manager and distributed storage system
  - Supports Haadop YARN
  - Can interface with HDFS, Amazon S3

- Originally from AMPLab, UC Berkeley