Elective exercise using Go and RPC

Corso di Sistemi Distribuiti e Cloud Computing
A.A. 2024/25

Valeria Cardellini

Laurea Magistrale in Ingegneria Informatica

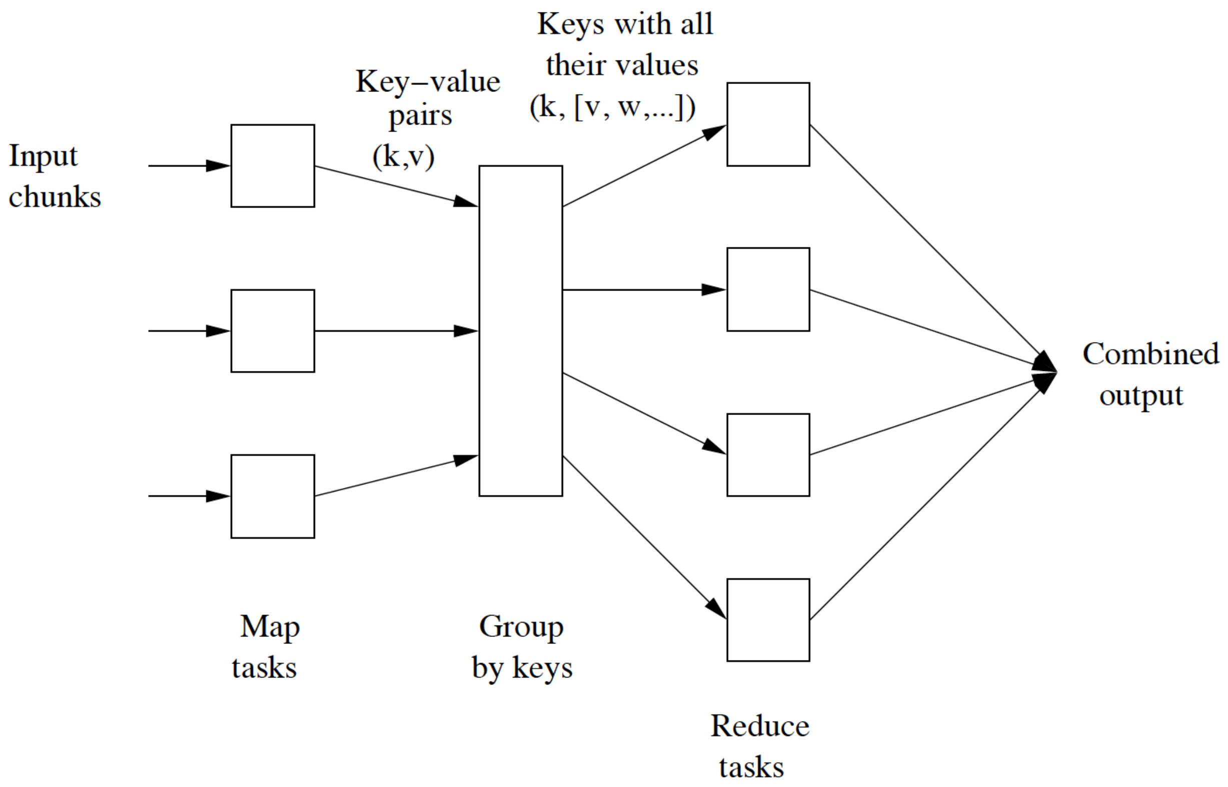
Elective exercise using Go and RPC

- Realize a distributed application that solves the sorting problem using MapReduce

> *The beauty of MapReduce is that any programmer can understand it, and its power comes from being able to harness thousands of computers behind that simple interface.*
>
> David Patterson

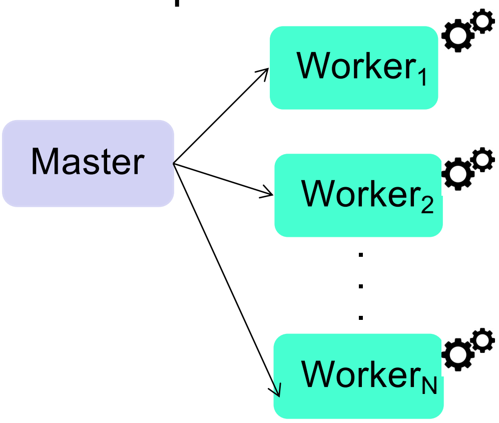
- Requirements:
  - Use either Go and RPC or Go and gRPC
  - Organize properly your code into separate files
  - 1 student per team (2 students: mandatory also one optional part)

MapReduce paradigm



Overview: architecture

- Exploit master-worker architecture
  - Distribute work among workers using RPC or gRPC for communication
- Master assigns map and reduce tasks (i.e., mapper and reducers) to workers
- Master divides input data into chunks and sends chunks to mappers by means of RPC
- Workers execute map and reduce tasks



Overview: distributed application

- Distribute work among mappers and reducers
- 3 phases of computation/communication
  - **Map**: each mapper sorts the chunk received from master
  - **Shuffle**: mappers send their intermediate result (sorted chunk) to reducers by partitioning their result among reducers
  - **Reduce**: each reducer merges partitions of intermediate results received from mappers, thus producing the sorted output and writing it on a file
- Exploit SPMD: mappers work in parallel, reducers work in parallel
- Need synchronization point (i.e., barrier) between mappers and reducers
  - No reducer can start until all mappers have completed their processing

Some simplifying assumptions

- Set of workers is known a priori (no discovery service is needed) and defined into a configuration file (IP addresses and ports)
- Master and workers can execute on same machine
  - IP address = localhost
- Can sort integer numbers (read from input file or randomly generated by master)
- Master and workers do not fail during computation

Optional

- Sample input data to optimize partitioning among reducers
  - Use a sorted list of sampled keys (*sample*) that defines the key range for each reducer
  - All keys such that *sample*[*i-1*] <= *key* < *sample*[*i*] are sent to reducer *i*
  - Inspired by TeraSort algorithm
- Containerized your distributed application
  - To build the image, see Go official image https://hub.docker.com/_/golang
  - Build a Docker container per application component and then use Docker Compose to orchestrate multiple containers on your machine

- 2 students per team: choose at least one option

Delivery

- When
  - By December 13, 2024
- What
  - Your code, including a README with instructions to run it
  - Optional: very short report describing the architecture of your distributed application and its main features
- How
  - By email
  - Use as mail subject: [SDCC] consegna esercizio in Go