Big Data Processing Engines

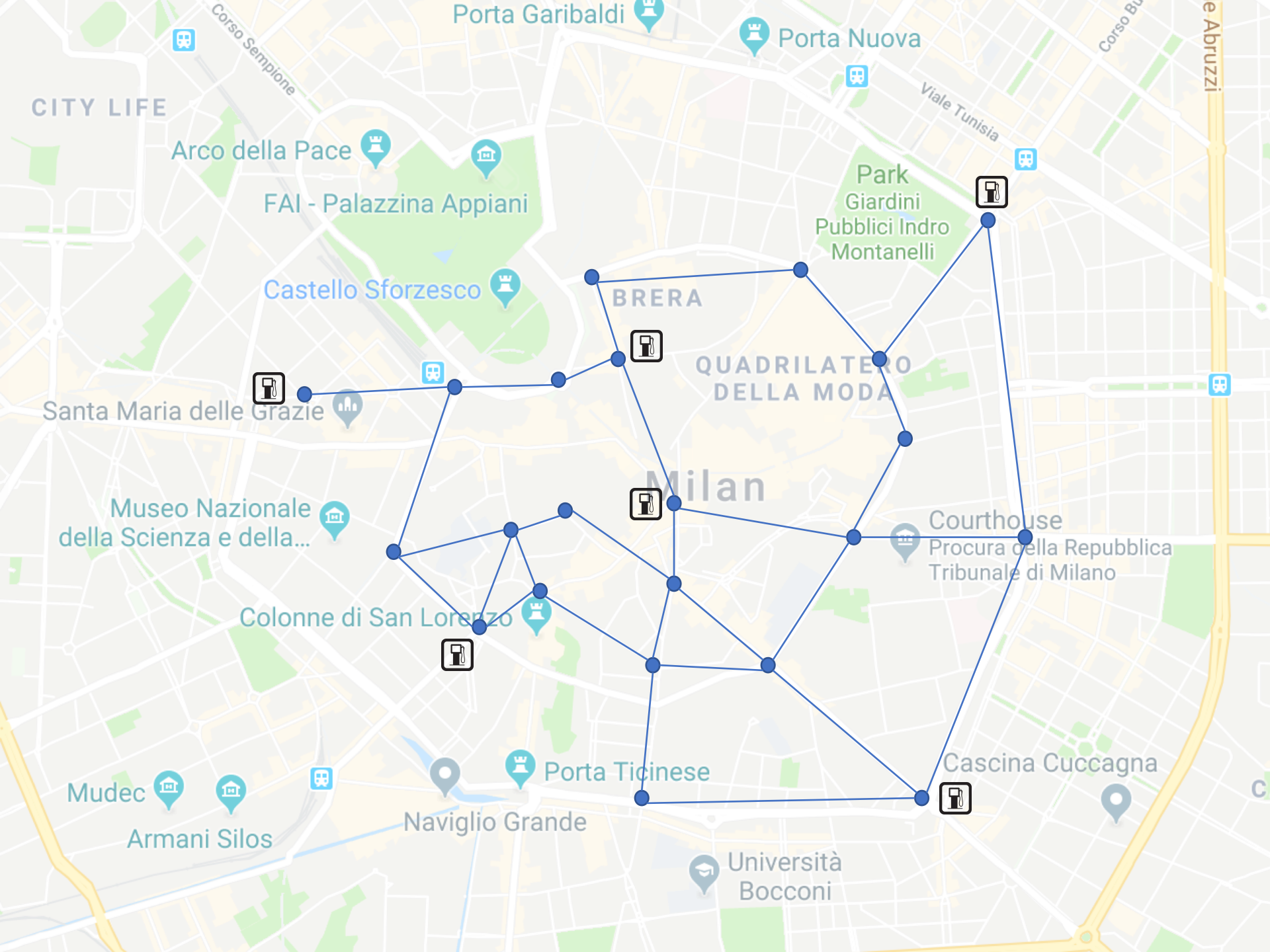
Alessandro Margara
alessandro.margara@polimi.it

License

This work is licensed under the Creative Commons Attribution-ShareAlike International Public License

Problem

- We have a map represented as a graph
  - Nodes are points of interest
  - Edges are roads
- We want to compute the shortest path from each point to the nearest gas station


Porta Garibaldi
Porta Nuova
Corso Sempione
Viale Tunisia
CITY LIFE
Arco della Pace
FAI - Palazzina Appiani
Park
Giardini
Pubblici Indro
Montanelli
Castello Sforzesco
BRERA
QUADRILATERO
DELLA MODA
Santa Maria delle Grazie
Milan
Museo Nazionale
della Scienza e della...
Courthouse
Procura della Repubblica
Tribunale di Milano
Colonne di San Lorenzo
Mudec
Armani Silos
Naviglio Grande
Porta Ticinese
Cascina Cuccagna
Università
Bocconi

Solutions?

- Compute the shortest path between each gas station and each node
  - This is the single-source shortest path problem
  - Well known algorithms (Dijkstra, Johnson, ...)
  - (If you are curious about the complexity: the best known algorithm works in O((N + E) log N), were N is the number of nodes and E is the number of edges, and we need to do that for each gas station)
- Then, for each node, we need to compare the distance computed from each gas station and keep only the minimum
  - E.g., we can store the best known solution for each node and update it as we discover better paths

Solutions?

- Possible optimization: we can discard nodes that are too far away
  - E.g., we assume that the maximum distance between any node and the nearest gas station is 50 km and we discard paths that are longer than that

Solutions?

- Did we ask ourselves the right questions?

- We implicitly made some strong assumptions
  - We can easily access the entire input data
  - We can easily store and update the state of the computation
    - Discovered paths, current minimum for each node, …
  - They fit in memory …
  - … or at least on the disk of a single node
    - Random access can then become a problem

Scalability

- In general, some problems only appear “at scale”
- Reading input and reading/storing the intermediate state of the computation can easily become the bottleneck
- If input/state do not fit in one machine, we need distributed solutions …
- … and then the coordination, communication can become the bottleneck

Scalability

- We need to consider
  - The scale of the problem
    - For now, let's focus on the volume of data only
  - The computing infrastructure we have

- Our story takes place in a datacenter of Google in the early 2000s

Assumptions
Scale of the problem

- Terabytes or more

- Data does not fit into a single disk
  - Or it is too expensive to read from a single disk
  - Sadly, there were no SSDs at that time …

- For sure does not fit in memory

Assumptions
Computing infrastructure

- Cluster of “normal” computers
- Hundreds to thousands of nodes
- No dedicated hardware
  - Less expensive
  - Easy to update frequently and incrementally
  - Not reliable! Hardware failures are common!

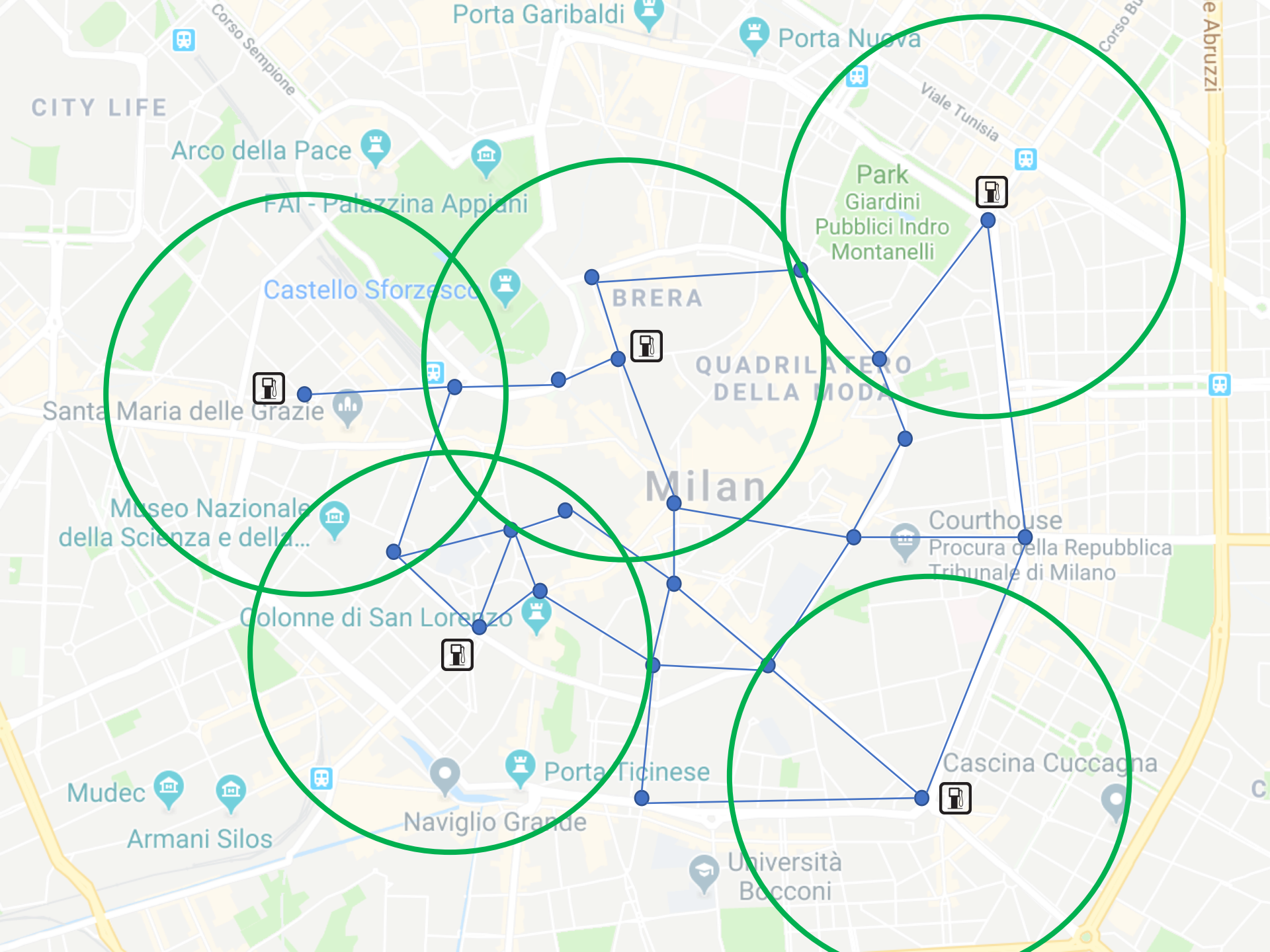
Back to the problem

- Now we have a clear picture of the assumptions …
- … better solutions?

Back to the problem

- Assuming that the maximum distance between any point and the nearest gas station is 50 km …

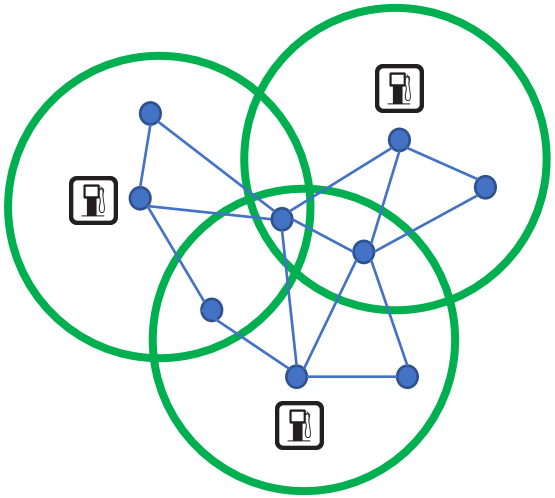
- … we can split the dataset into blocks
  - Each block b_i centered around gas station i
  - Different blocks can be stored on different computers
  - For each block b_i, we can compute the distance between gas station i and any node in the block
    - The computation is independent for each block
    - Blocks can be processed entirely in parallel


Porta Garibaldi
Porta Nuova
Corso Sempione
Viale Tunisia
CITY LIFE
Arco della Pace
FAI - Palazzina Appiani
Park
Giardini
Pubblici Indro
Montanelli
Castello Sforzesco
BRERA
QUADRILATERO
DELLA MODA
Santa Maria delle Grazie
Milan
Museo Nazionale
della Scienza e della...
Courthouse
Procura della Repubblica
Tribunale di Milano
Colonne di San Lorenzo
Mudec
Armani Silos
Naviglio Grande
Porta Ticinese
Cascina Cuccagna
Università
Bocconi

Back to the problem

- Next, we need to determine the closest gas station and the shortest path for each node

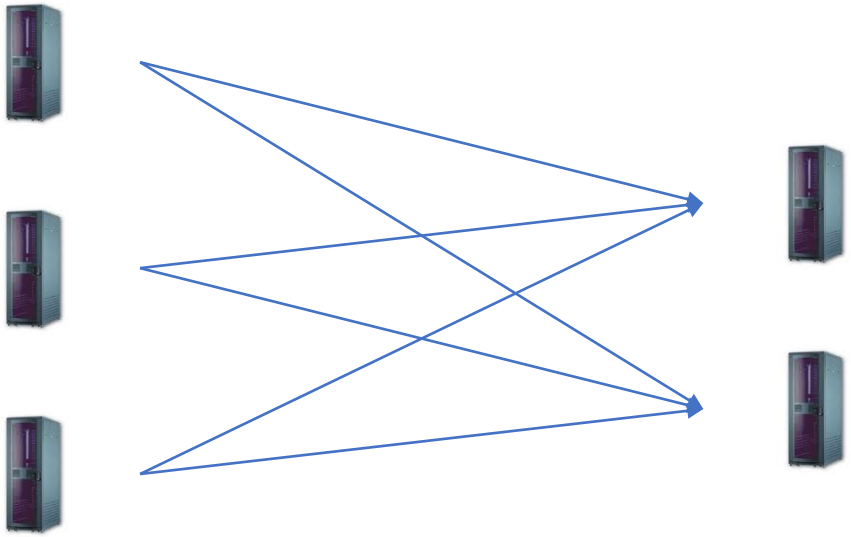
- This requires communication, because a point can be at the intersection of several blocks



Back to the problem

- We can repartition the data by node
- Again, the shortest path (minimum distance) can be computed independently for each node
  - Each node can be processed in parallel on a different machine

The solution



Computation of paths

- Data split by gas station
- Each gas station can be processed in parallel
- Nodes can read input blocks in parallel

Computation of shortest paths

- Data split by node
- Each node can be processed in parallel

Open questions

- How to optimize the access to input data?
- How to optimize the reordering of information from the first to the second phase
- How to deal with (hardware) failures?

MapReduce

- Introduced by Google in 2004
- Programming model and framework
- Adopts the approach we have seen in the previous slides
- Provides a solution to the open problems we listed

MapReduce Programming model

- Fixed structure

- Computations consist of two phases (map, reduce)
  - Developers need only specify a map and a reduce function
  - Map: computes solutions for a portion of the input data
    - Paths for a given gas station
  - Reduce: combines/aggregates solutions for a given key
    - The key is the node
    - Take the minimum path

MapReduce Framework

- The framework (automagically) takes care of all the aspects related to
  - Scheduling: allocates resources for mappers and reducers
  - Data distribution: moves data from mappers to reducers
    - Actually, deploys mappers and reducers close to their input data
  - Fault tolerance: transparently handles the crash of one or more nodes

MapReduce: scheduling

- Input, output, intermediate data stored on a distributed filesystem (GFS)

- One master, many workers
  - Input data split into M map tasks
  - Reduce phase partitioned into R reduce tasks
  - Tasks are assigned to workers dynamically

- Master assigns each map task to a free worker
  - Considers locality of data to worker when assigning a task
  - Worker reads task input (often from local disk)
  - Worker produces local files containing intermediate k/v pairs

- Master assigns each reduce task to a free worker
  - Worker reads intermediate k/v pairs from map workers
  - Worker sorts & applies user's reduce operation to produce the output

MapReduce: data locality

- Goal: conserve network bandwidth
- In GFS, data files are divided into 64MB blocks and 3 copies of each are stored on different machines
- Master program schedules map() tasks based on the location of these replicas
  - Put map() tasks physically on the same machine as one of the input replicas (or, at least on the same rack / network switch)
- This way, thousands of machines can read input at local disk speed
  - Otherwise, rack switches would limit read rate

MapReduce: fault tolerance

- On worker failure
  - Master detects failure via periodic heartbeats
  - Both completed and in-progress map tasks on that worker should be re-executed
    - Output stored on local disk
  - Only in-progress reduce tasks on that worker should be re-executed
    - Output stored in global file system
  - All reduce workers will be notified about any map re-executions
- On master failure
  - State is check-pointed to GFS: new master recovers & continues

MapReduce: stragglers

- Stragglers = tasks that take long time to execute
  - Slow hardware, poor partitioning, bug, …
- When done with most tasks …
- … reschedule any remaining executing task
  - Keep track of redundant executions
  - Significantly reduces overall run time

MapReduce: usage

- Typical MapReduce application
  - Sequence of steps, each requiring map & reduce
  - Series of data transformations
  - Iterating until reach convergence
    - E.g., Google PageRank

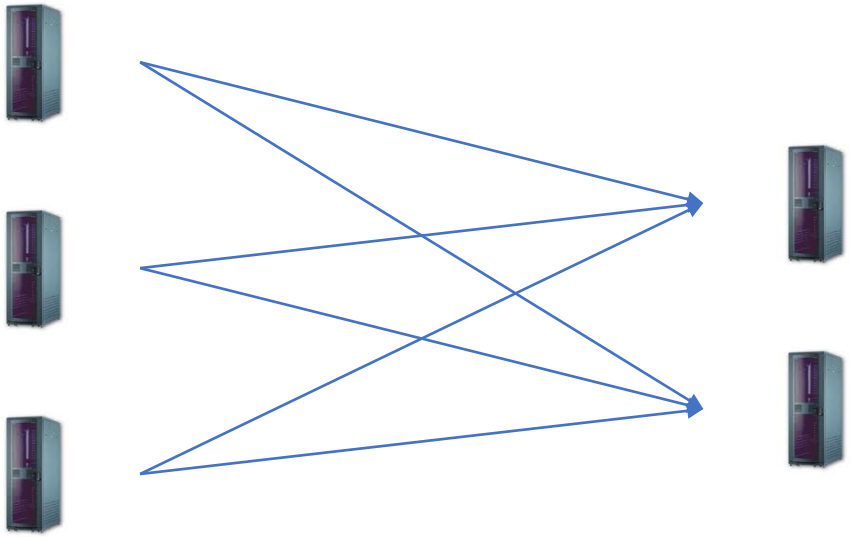
Lessons learned

- MapReduce is considered the first Big Data processing platform

- Why is it so important?
- What is the core idea it introduced?

- Let’s go back to the solution we provided for the problem

The solution



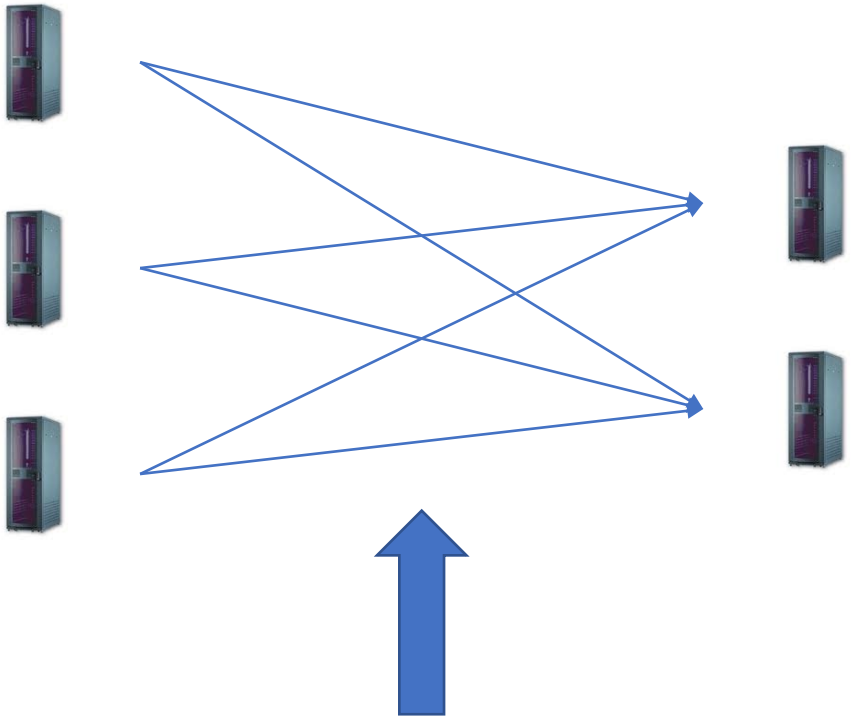
Computation of paths

- Data split by gas station
- Each gas station can be processed in parallel
- Nodes can read input blocks in parallel

Computation of shortest paths

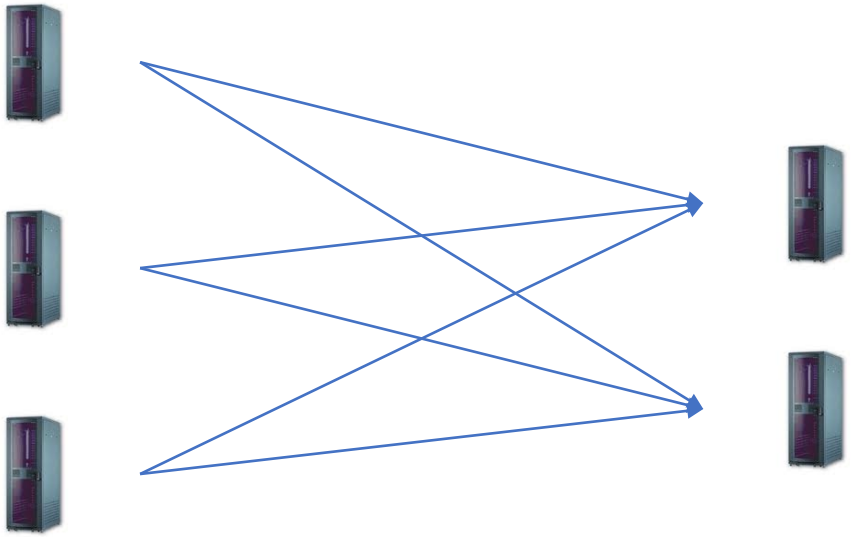
- Data split by node
- Each node can be processed in parallel

The solution



From (global) *state* to (streaming) *data*
From mutable *state* to immutable *data transformations*
A functional approach!

The solution



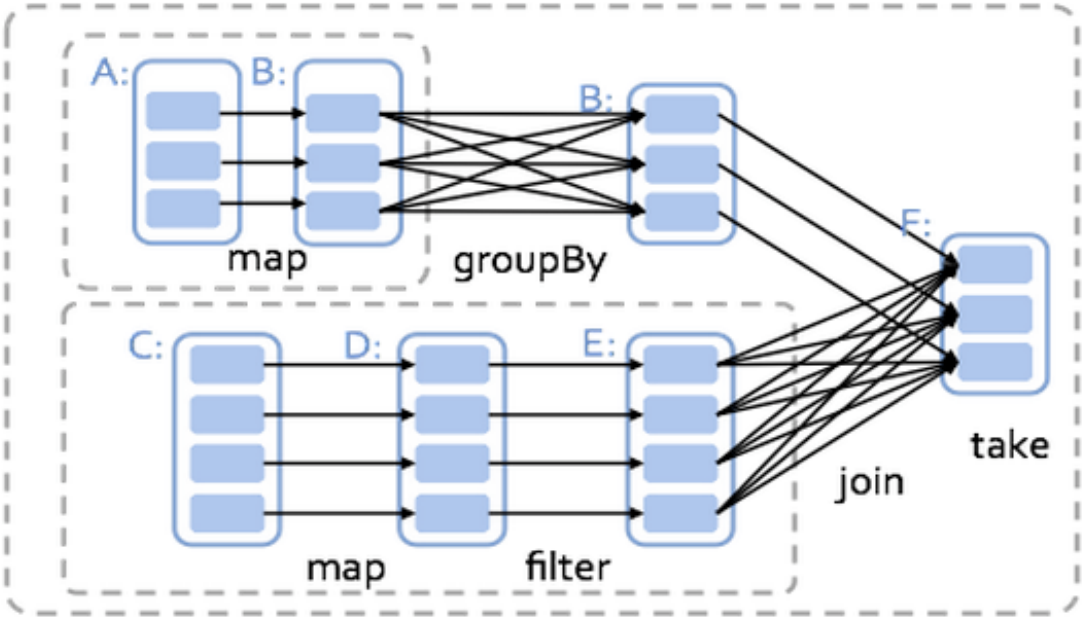
Different channels

- Distributed filesystem
- Messages (e.g., TCP)
- Queues (e.g., Kafka)

Evolution: Apache Spark

- From two to multiple phases
- Many more functions
  - Filter, FlatMap, Join, …
- Iterative computations
  - Repeat some some operations until some conditions hold
  - Useful for statistical analysis / machine learning tasks that refine partial solutions to reach the desired level of accuracy
- From disk-based to in-memory
  - Each phase caches intermediate data in cache

Evolution: Apache Spark



Evolution: Spark Streaming

- So, Big Data processing systems are (conceptually) streaming data from operator to operator
  - In fact, operators are scheduled on the cluster dynamically when there is data available for them
- Idea: use them to process data that is actually streaming
  - Velocity dimension
- Spark Streaming: split the streaming data into small blocks (micro-batches) and run the computation for each of them independently
  - Suitable if delays in the order of seconds are acceptable

Evolution: Apache Flink

- Apache Flink takes the opposite approach
- Instead of building streaming computations on top of batch computations …
- … it considers batch computations as a special case of streaming computations
  - Bounded datasets as a special case of (unbounded) streams

Evolution: Apache Flink

- Change in architecture
- Operators are not dynamically scheduled
  - They always remain up and running
- Operators do not write intermediate values to disk
  - They communicate through messages
- Different approach to fault tolerance
  - Periodic distributed snapshot of the state of operators
  - Replay all input elements that are not part of the snapshot

Questions?