Distributed Systems

Monsoon 2024

Lecture 9

International Institute of Information Technology

Hyderabad, India

Slides credit to Dr. Yogesh Simmhan, IISc,
and other sources mentioned.

Logistics

- Extended the deadline for HW 3 to 12 midnight, Monday, September 02, 2024.

Distributed Computing Platforms

- We will study yet another distributed computing platform that has gained recent popularity.
- We will study what this platform and its recent avatars offer.

Distributed Computing

- Distributed Computing involves
  - Clusters of machines connected over network
  - Distributed Storage
    - Disks attached to clusters of machines
    - Network Attached Storage

- **Commodity** clusters
  - Commodity: Available off the shelf at large volumes
  - Lower Cost of Acquisition
  - Cost vs. Performance
  - Low disk bandwidth, and high network latency
  - CPUs typically comparable

Distributed Computing

- Distributed Computing involves
  - Clusters of machines connected over network
  - Distributed Storage
    - Disks attached to clusters of machines
    - Network Attached Storage

How can we make effective use of multiple machines?

- **Commodity** clusters
  - Commodity: Available off the shelf at large volumes
  - Lower Cost of Acquisition
  - Cost vs. Performance
  - Low disk bandwidth, and high network latency
  - CPUs typically comparable

How can we use many of such machines of modest capability?

However,

- Commodity clusters have lower reliability
  - Mass-produced
  - Cheaper materials
  - Smaller lifetime (~3 years)
- *How can applications easily deal with failures?*
- *How can we ensure availability in the presence of faults?*
- *While at the same time…*

While at the same time,

- Realize that programming distributed systems is difficult
  - Divide a job into multiple tasks
  - Understand dependencies between tasks: Control, Data
  - Coordinate and synchronize execution of tasks
  - Pass information between tasks
  - Avoid race conditions, deadlocks
- Paralel and distributed programming models/ languages/ abstractions/platforms try to make these easy
  - E.g. Assembly programming vs. C++ programming
  - E.g. C++ programming vs. Matlab programming

Distributed Programming

- Recall that distributed systems are characterized as a collection of autonomous computers communicating over a network of links.
- Some typical features of such a system are:
  - No common physical clock
    - Clocks can drift and no notion of a common clock.
    - Systems can be asynchronous too.

Distributed Programming

- Recall that distributed systems are characterized as a collection of autonomous computers communicating over a network of links.
- Some typical features of such a system are:
  - No common physical clock
    - Clocks can drift and no notion of a common clock.
    - Systems can be asynchronous too.
  - No shared memory
    - Use messages for communication along with their semantics.

Distributed Programming

- Recall that distributed systems are characterized as a collection of autonomous computers communicating over a network of links.
- Some typical features of such a system are:
  - No common physical clock
    - Clocks can drift and no notion of a common clock.
    - Systems can be asynchronous too.
  - No shared memory
    - Use messages for communication along with their semantics.
  - Geographical separation
    - Can allow for wider scope of operations, e.g., SETI

Distributed Programming

- Recall that distributed systems are characterized as a collection of autonomous computers communicating over a network of links.
- Some typical features of such a system are:
  - No common physical clock
    - Clocks can drift and no notion of a common clock.
    - Systems can be asynchronous too.
  - No shared memory
    - Use messages for communication along with their semantics.
  - Geographical separation
    - Can allow for wider scope of operations, e.g., SETI
  - Autonomy
    - Processors are loosely coupled but cooperate with each other
  - Heterogeneity
    - All processors need not be alike.

Challenges of Distributed Programming

- Distributed programming is inherently challenging.
- Need to ensure that several geographically separate computers collaborate
  - efficiently,
  - reliably,
  - transparently, and
  - scalable manner
- One can also talk of inherent complexity and accidental complexity [Check D. Schmidt, VU]

Challenges of Distributed Programming

- One can also talk of inherent complexity and accidental complexity [Check D. Schmidt, VU]
- Inherent complexity due to
  - Latency: Computers can be far apart
  - Reliability: Recovering from failures of individual or collection of computers
  - Partitioning: Sometimes, failures can partition the system.
  - Ordering: Lack of global clock means message/event ordering is difficult
  - Security: Program as well as computational aspects

Challenges of Distributed Programming

- Distributed programming is inherently challenging.
- Need to ensure that several geographically separate computers collaborate
  - Efficiently, reliably, transparently, and scalable manner
- One can also talk of inherent complexity and accidental complexity [Check D. Schmidt, VU]
- Inherent complexity due to
  - Latency, Reliability, Partitioning, Ordering, Security
- Accidental complexity due to
  - Low-level APIs: Not enough abstraction
  - Poor debugging tools: Poor fault location
  - Algorithmic decomposition: Choice of algorithmic techniques
  - Continuous re-invention: Key components can change!

Enter Map-Reduce

- **MapReduce** is a distributed data-parallel programming model from Google
    - Introduced close to 20 years ago.
- MapReduce works best with a distributed file system, called **Google File System (GFS)**
- **Hadoop** is the open source framework implementation from Apache that can execute the MapReduce programming model
- **Hadoop Distributed File System (HDFS)** is the open source implementation of the GFS design
- Amazon's PaaS is yet another implementation of Map-Reduce

Map-Reduce

"A simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs."

Dean and Ghermawat, "MapReduce: Simplified Data Processing on Large Clusters", OSDI, 2004

A High Level View

- Map Reduce provides
  - Clean abstraction for programmers
  - Automatic parallelization & distribution
  - Fault-tolerance
  - A batch data processing system
  - Status and monitoring tools

Map-Reduce Vs. RDBMS

- Why not RDBMS – the old data workhorse?
  - RDBMS suffer from a huge seek time resulting in huge access times.
- Map-Reduce comes with the implicit assumption that nearly the entire dataset is relevant for each query.
  - MapReduce is therefore akin to a batch query processor.
- MapReduce is a good fit for problems that need to analyze the whole dataset, in a batch fashion, particularly for ad hoc analysis.

Map-Reduce Vs. RDBMS

- An RDBMS is good for point queries or updates, where the dataset has been indexed to deliver low-latency retrieval and update times of a relatively small amount of data.
- MapReduce suits applications where the data is written once, and read many times, whereas a relational database is good for datasets that are continually updated.
- MapReduce works well on unstructured or semistructured data

Map-Reduce Vs. RDBMS

- Relational data is often normalized to retain its integrity and remove redundancy.
- Normalization poses problems for MapReduce,
  - reading a record a nonlocal operation,
  - one of the central assumptions that MapReduce makes is that it is possible to perform (high-speed) streaming reads and writes.
- A web server log is a good example of a set of records that is not normalized
  - the client hostnames are specified in full each time, even though the same client may appear many times
  - Hence, logfiles are particularly well-suited to analysis with MapReduce

Map-Reduce Vs. RDBMS

- Over time, however, the differences between relational databases and MapReduce systems are reducing.
- Relational databases starting to incorporate some of the ideas from MapReduce
  - Aster Data’s and Greenplum’s databases
- Higher-level query languages built on MapReduce (such as Pig and Hive) make MapReduce systems more approachable to traditional database programmers.

Map-Reduce

- MapReduce might sound like quite a restrictive programming model
- Mappers and reducers run with very limited coordination between one another.
- Typical problems that we can use Map-Reduce programming model are from image analysis, to graph-based problems, to machine learning algorithms.
- It can’t solve every problem, of course, but it is a general data-processing tool

MapReduce: Data-parallel Programming Model

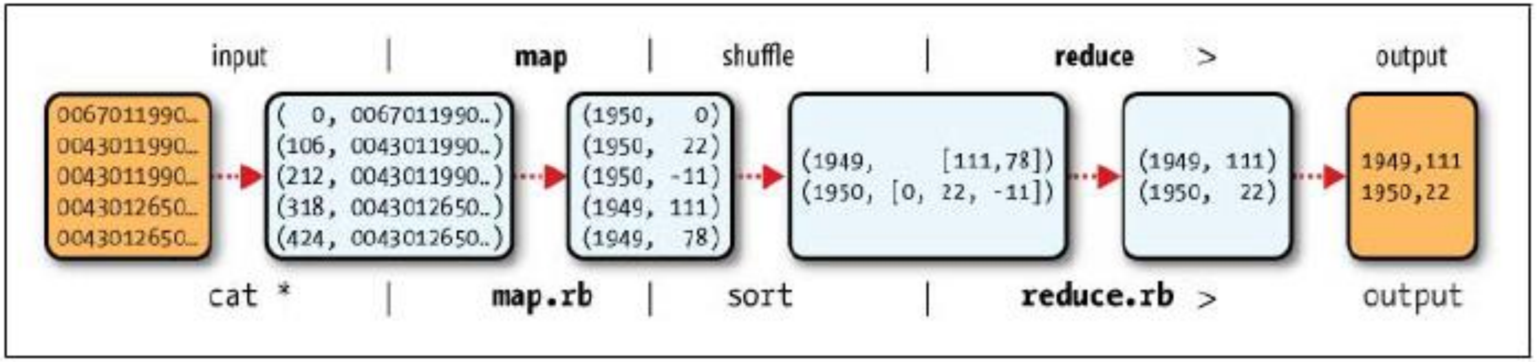
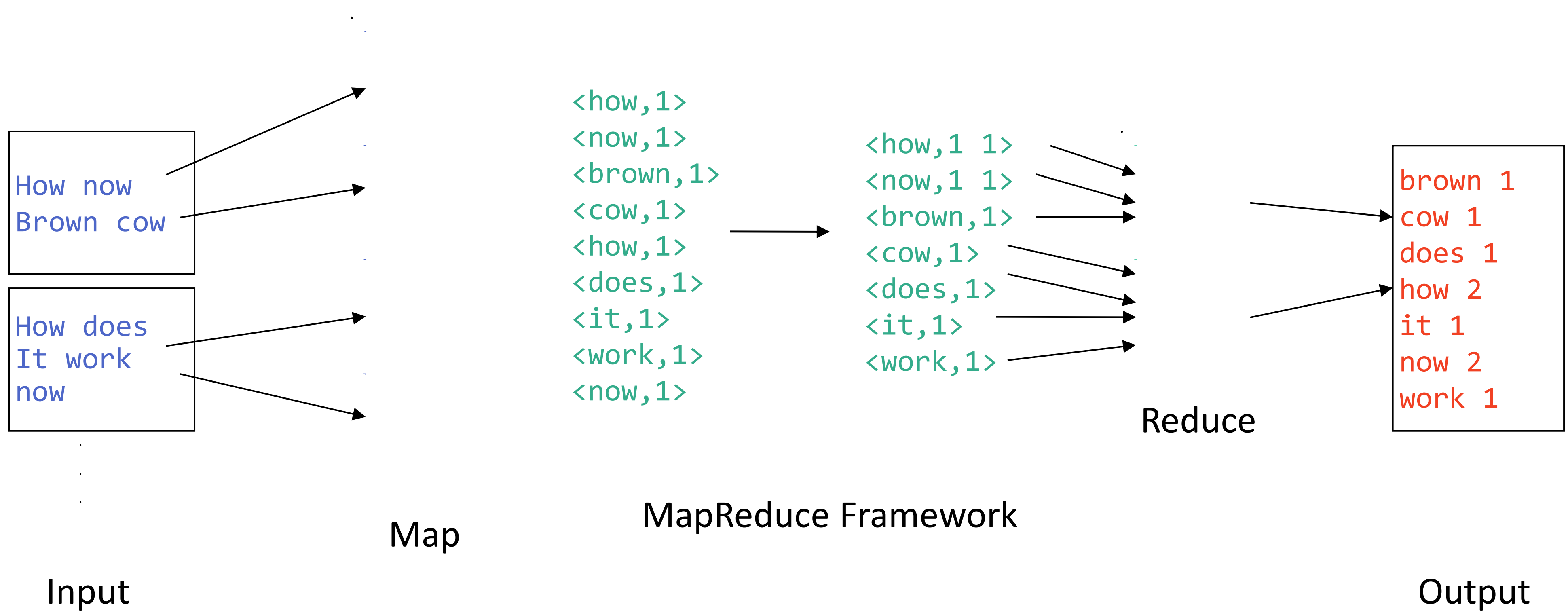


Figure 2-1. MapReduce logical data flow

- Process data using map & reduce functions
- $\texttt{map}(k_i, v_i) \rightarrow \texttt{List}\langle k_m, v_m \rangle[\,]$
  - ***map*** is called on every input item
  - Emits a series of intermediate key/value pairs
- All values with a given key are **grouped** together
- $\texttt{reduce}(k_m, \texttt{List}\langle v_m \rangle[\,]) \rightarrow \texttt{List}\langle k_r, v_r \rangle[\,]$
  - ***reduce*** is called on every unique key & all its values
  - Emits a value that is added to the output

MapReduce: Word Count

Map(k1,v1) → list(k2,v2)
Reduce(k2, list(v2)) → list(k2,v2)



Distributed Word Count

Map

- Input records from the data source
  - lines out of files, rows of a database, etc.
- Passed to map function as key-value pairs
  - Line number, line value
- map() produces *zero or more intermediate values*, each associated with an output key.

Map

- **Example: Upper-case Mapper**

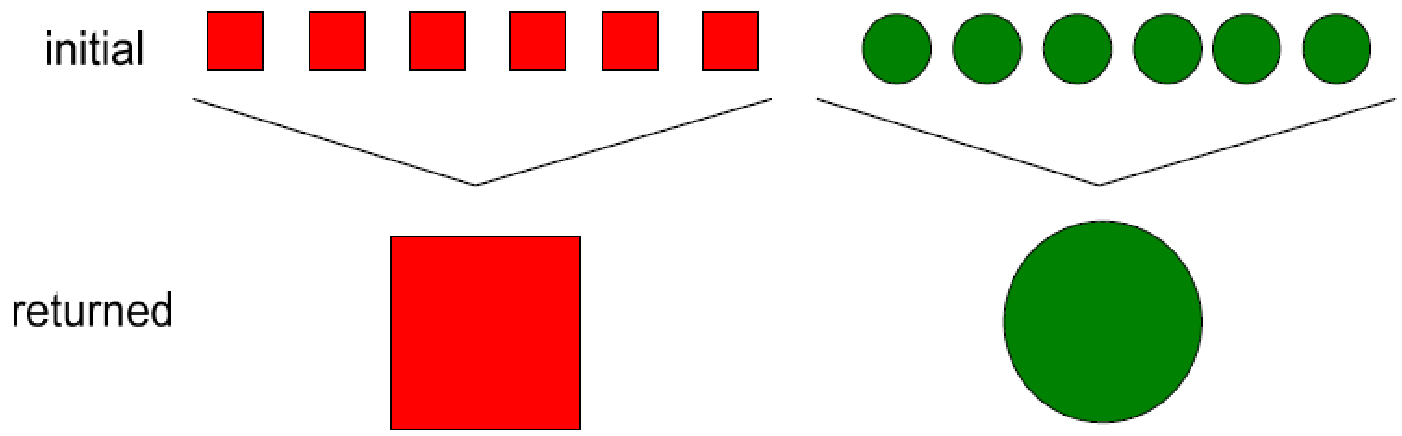
```
map(k, v) { emit(k.toUpper(), v.toUpper()); }

("foo", "bar") → ("FOO", "BAR")
("Foo", "other") → ("FOO", "OTHER")
("key2", "data") → ("KEY2", "DATA")
```

- **Example: Filter Mapper**

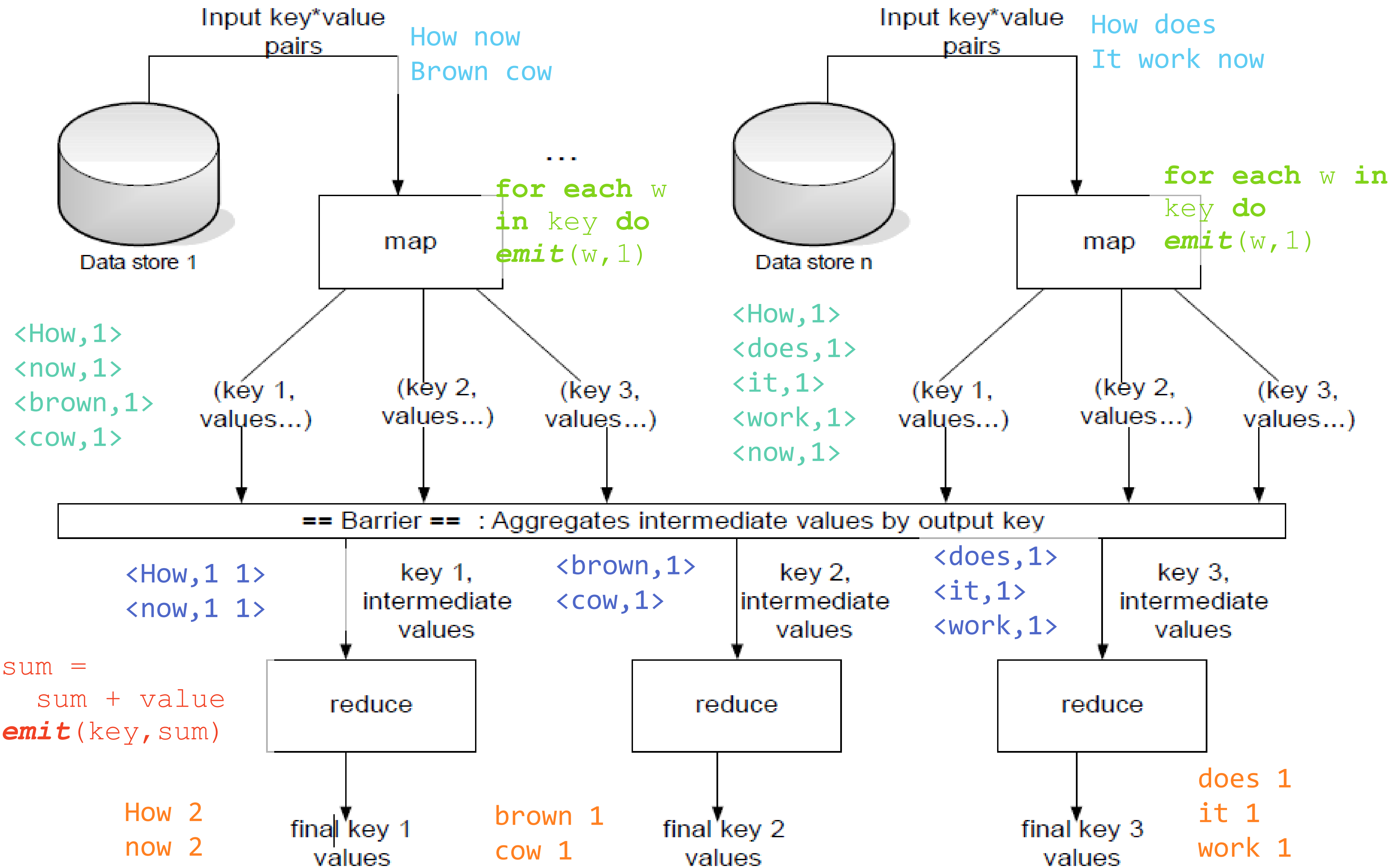
```
map(k, v) { if (isPrime(v)) then emit(k, v); }

("foo", 7) → ("foo", 7)
("test", 10) → () //nothing emitted
```

Reduce

- All the intermediate values from map for a given output key are combined together into a list
- reduce() combines these intermediate values into one or more final values for that same output key … Usually one final value per key
- One output “file” per reducer



MapReduce: Word Count Drilldown



Mapper/Reducer Tasks *vs.* Map/Reduce Methods

- Number of Mapper and Reducer tasks is specified by user
- Each **Mapper/Reducer** **task** can make multiple calls to **Map/Reduce** **method**, sequentially
- Mapper and Reducer tasks may run on different machines
- Implementation framework decides
  - Placement of Mapper and Reducer tasks on machines
  - Keys assigned to mapper and reducer tasks
  - But can be controlled by user…

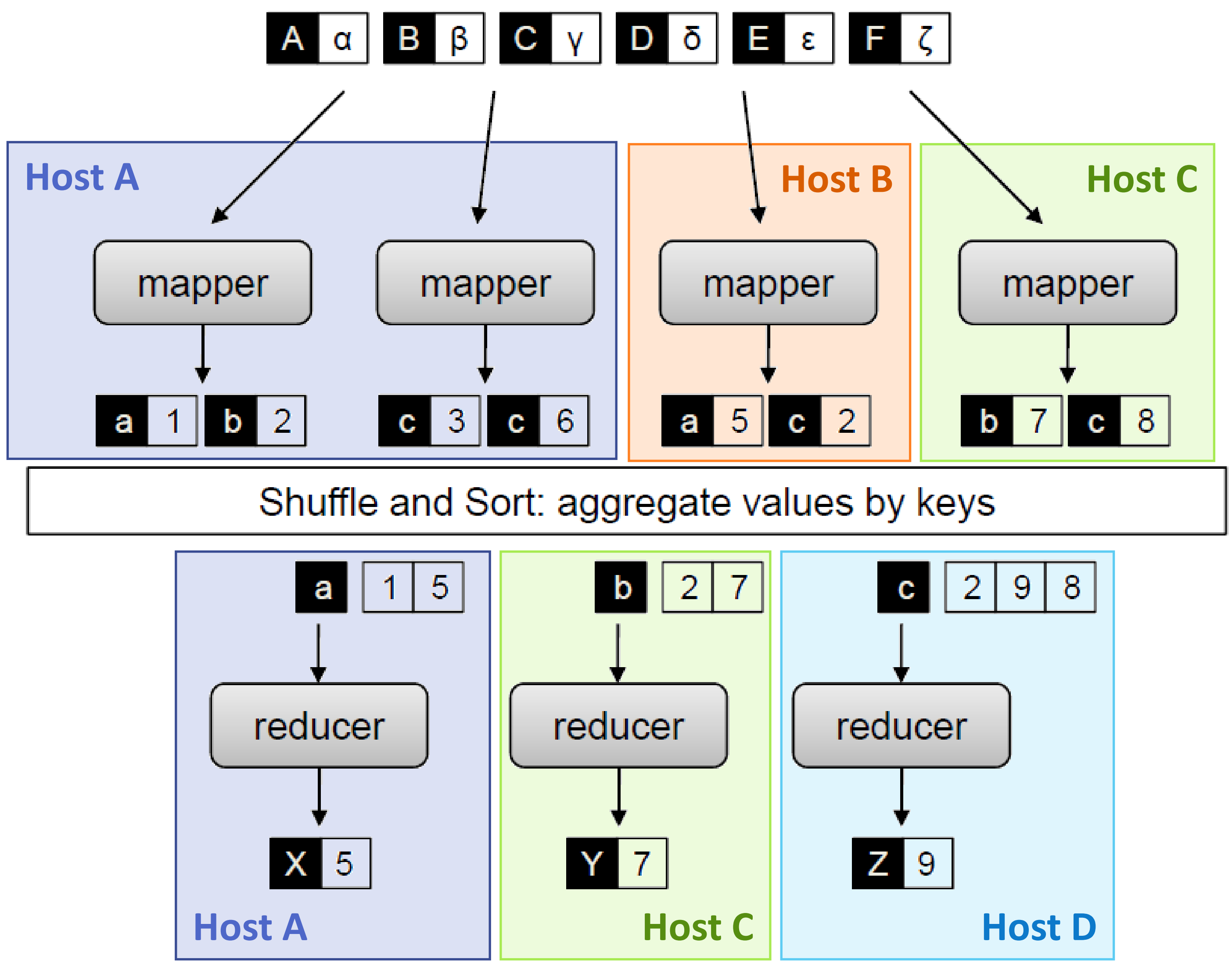
Shuffle & Sort: *The Magic happens here!*

- **Shuffle** does a "group by" of keys from all mappers
  - Similar to SQL groupBy operation

- **Sort** of *local keys* to *Reducer task* performed
  - Keys arriving at each reducer are sorted
  - No sorting guarantee of keys across reducer tasks
- No ordering guarantees of values for a key
  - Implementation dependent

- Shuffle and Sort *implemented efficiently* by framework

Map-*Shuffle*-*Sort*-**Reduce**



Data-Intensive Text Processing with MapReduce, Jimmy Lin, 2010

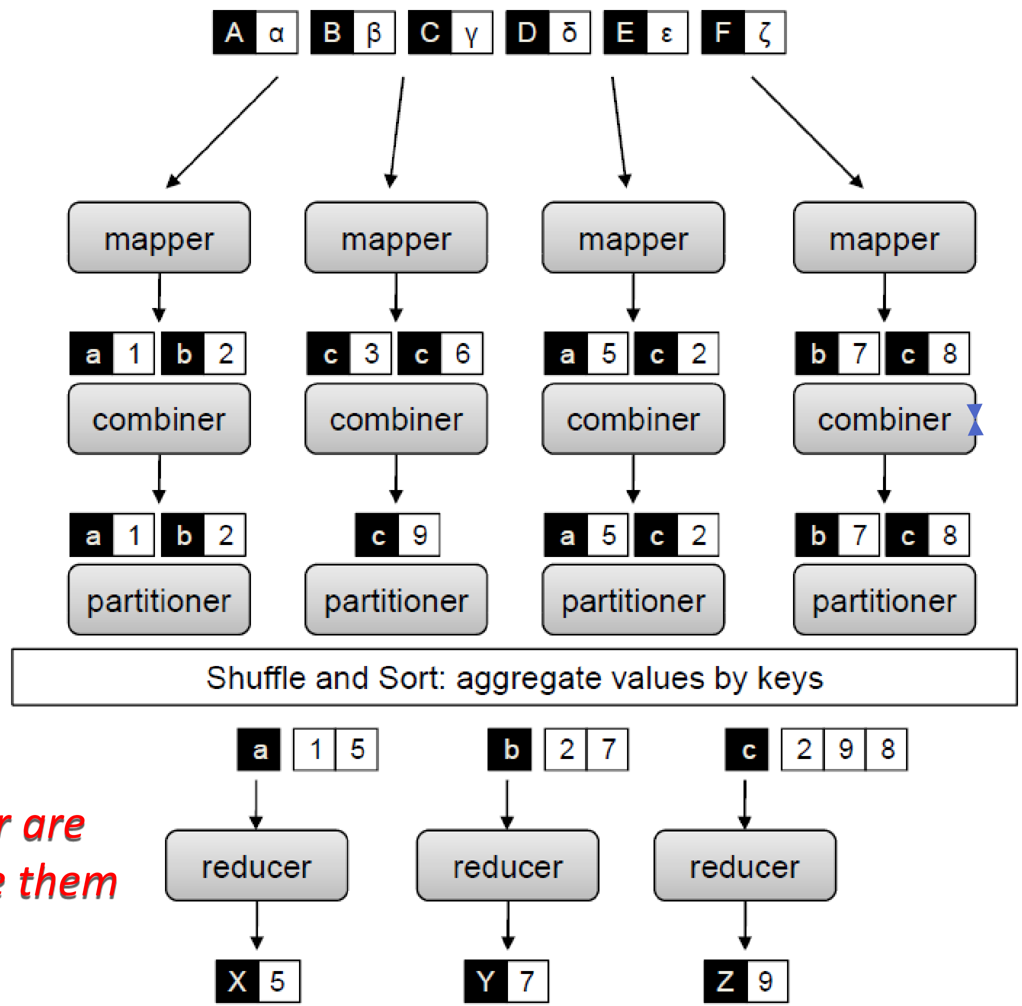
Optimization: **Combiner**

- Logic runs on output of Map tasks, on the map machines
  - "Mini-Reduce," only on local Map output
- Output of Combiner sent to shuffle
  - Saves bandwidth before sending data to Reducers
- Same *input* and *output types* as Map's *output* type
  - `Map(k,v)` → `(k',v')`
  - `Combine(k',v'[])` → `(k',v')`
  - `Reduce(k',v'[])` → `(k'',v'')`

Optimization: **Partitioner**

- Decides assignment of intermediate keys grouped to specific Reducer tasks
  - Affects the load on each reducer task
- Sorting of local keys for Reducer task done after partitioning
- Default is hash partitioning
  - **`HashPartitioner(key, nParts)`** → **`part`**
  - Number of Reducer (**nParts**) tasks known in advance
  - Returns a partition number [0, nParts)
  - Default partitioner balances number of keys per Reducer … *assuming uniform key distribution*
  - May not balance the number of values processed by a Reducer

Map-*MiniShuffle-Combine-Partition*-Shuffle-Sort-**Reduce**



Combine & Partition phases could be interchanged, based on implementation

Combiner & Partitioner are powerful constructs. Use them wisely!

Data-Intensive Text Processing with MapReduce, Jimmy Lin, 2010

MapReduce for Histogram

7	2	11	2
2	1	11	4
9	10	6	6
6	3	2	8
0	5	1	10
2	4	8	11
5	0	1	0

M		M	
1,1	0,1	2,1	0,1
0,1	0,1	2,1	1,1
2,1	2,1	1,1	1,1
1,1	0,1	0,1	2,1
0,1	1,1	0,1	2,1
0,1	1,1	2,1	2,1
1,1	0,1	0,1	0,1

Shuffle

2,1	0,1	0,1	1,1
2,1	0,1	0,1	1,1
2,1	0,1	0,1	1,1
2,1	0,1	0,1	1,1
2,1	0,1	0,1	1,1
2,1	0,1	0,1	1,1
2,1			1,1
2,1			1,1
2,8	0,12		1,8

Data transfer & *shuffle* between Map & Reduce **(28 items)**

```
int bucketWidth = 4 // input

Map(k, v) {
emit(floor(v/bucketWidth), 1)
// <bucketID, 1>
}

// one reduce per bucketID
Reduce(k, v[]){
sum=0;
foreach(n in v[])  sum++;
emit(k, sum)
// <bucketID, frequency>
}
```

Combiner Advantage

- Mini-Shuffle lowers the overall cost for Shuffle
- E.g. *n* total items emitted from *m* mappers
- Reduces network transfer and Disk IO costs
  - In ideal case, `m` items vs. n items *written and read from disk, transferred over network* (`m<<n`)
- Shuffle, less of an impact
  - If more mapper tasks are present than reducers, higher parallelism for doing groupby and mapper-side partial sort.
  - Local Sort on reducer is based on number of unique keys, which does not change due to combiner.

MapReduce: Recap

- Programmers must specify:

map (k, v) → <k', v'>*

reduce (k', v'[]) → <k'', v''>*

All values with the same key are reduced together

- Optionally, also:

partition (k', number of partitions) → partition for k'

Often a simple hash of the key, e.g., hash(k') mod n

Divides up key space for parallel reduce operations

combine (k', v') → <k', v'>*

  - Mini-reducers that run in memory after the map phase
  - Used as an optimization to reduce network traffic

- The execution framework handles *everything else*…

What Does it Solve?

- Scalable data analytics
- Data not viewed or constrained by disk read/writes but find meaning in data via computation over sets of keys and values.
- Abstract out lots of programming details.
- Forgo removing data redundancy via normalization of tables.
  - The penalty of normalization is reading a record requires reading from multiple sources/tables/files.

“Everything Else”

- The execution framework handles everything else…
  - Scheduling: assigns workers to map and reduce tasks
  - “Data distribution”: moves processes to data
  - Synchronization: gathers, sorts, and shuffles intermediate data
  - Errors and faults: detects worker failures and restarts
- Limited control over data and execution flow
  - All algorithms must expressed in m, r, c, p
- You don’t know:
  - Where mappers and reducers run
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which intermediate key a particular reducer is processing

Map-Reduce

- Failures are handled via the jobtracker and the tasktracker.
- The tasktracker notices failures of tasks including
  - Runtime exceptions
  - Sudden exit of the code executing the task
  - Tasks hanging – observed via timeouts
- If the jobtracker itself fails, Map-Reduce has no mechanisms to save and restore computations.

Other Variants of Map-Reduce

- YARN – Yet Another Resource Negotiator
- Developed to address the scalability concerns of Map-Reduce once the number of nodes is of the order of 10^4 or more.
- YARN splits the responsibilities of the jobtracker into two separate entities
  - a resource manager to manage the use of resources across the cluster, and
  - an application master to manage the lifecycle of applications running on the cluster.
- YARN has more sophisticated mechanisms to save state of the jobtracker on failure.
- YARN is developed to be more general than MapReduce,
  - In fact MapReduce can be seen as just a type of YARN application

Further Reading and Action Items

- Several piece-meal tutorials available online.
- A one-place material available as online book, Hadoop: The Definitive Guide, by Tom White.
  - Will post this PDF in our course moodle.
  - Read chapters 1, 2, 3, and 6.
- Homework 4 that gives you practice on Map-Reduce will be posted soon.