CS4225/CS5425 Big Data Systems for Data Science

Revision (1st Half)

Bingsheng He
School of Computing
National University of Singapore
hebs@comp.nus.edu.sg

School of Computing

Exam Info

- Time: 29 APR 2026 09:00-11:00
  - Students are allowed to enter the venue 10 mins before exam starts.
  - Students will not be permitted to enter the venue after 1 hour from the start of the exam.
- Venue: TBD
- F2F - Hardcopy (pen and paper)
- Open Book Exam
  - Any physical materials are allowed
  - Calculator is allowed
  - Any other electronics devices are NOT allowed

Exam

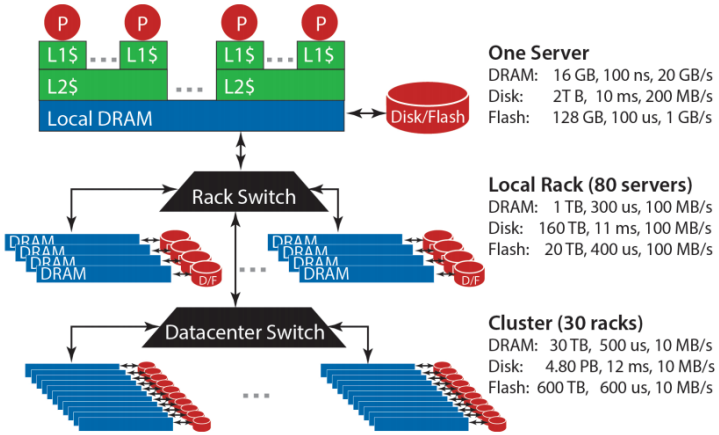
- Focus is on understanding and application, not facts / memorization
- Question structures (total 50 marks):
  - Ture / False question with a short explanation / justification
  - Application / Scenario Based Question
    - Give you a practical scenario and let you come out a solution / suggestion
- Example questions
  - Integrative: Require you to combine knowledge from different weeks of content
  - "Application": Require you to apply your knowledge of fundamental concepts to reasonably practical scenarios.
  - "Why not": Example, Tommy proposed a solution A to solve problem B. Tell me what is the problem with solution A and how to overcome this problem

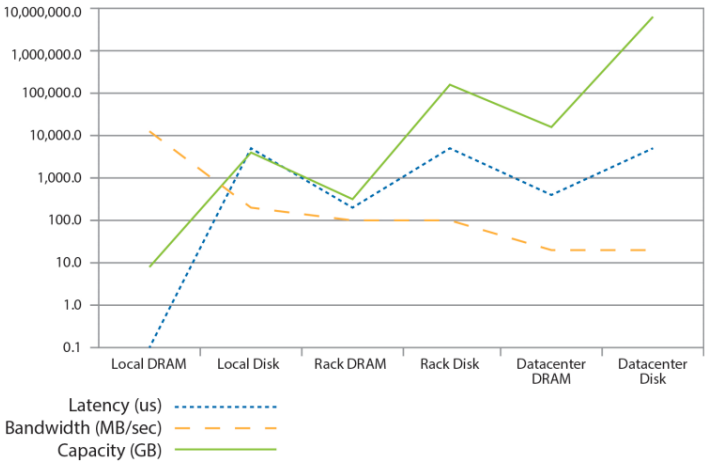
Scope of Exam

- Scope: content discussed in the lecture which appears in slides
- Out of scope:
  - Content only found in the notes underneath the slides (these can be ignored)
  - Content only discussed in response to student questions (i.e. beyond the scope of the slides)
  - Content marked with “Details” or that I mentioned is out of scope
  - If there are any tasks which ask for code, they will allow pseudocode. As long as your pseudocode is reasonable / understandable by the grader, we will accept it.
  - Historical details
  - Further readings
  - Supplementary slides

Infrastructure for Big Data

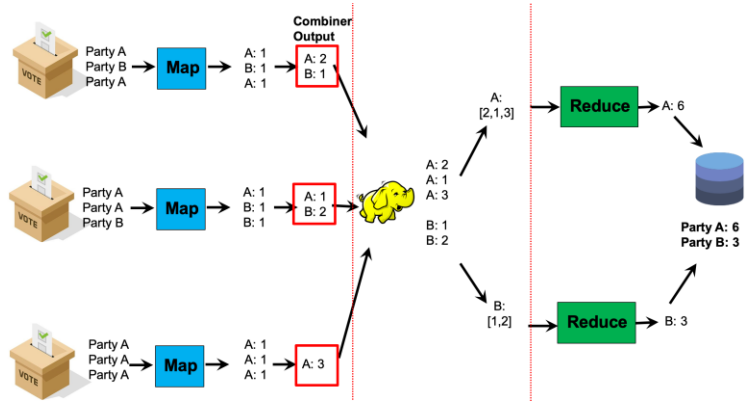
- Data center architecture
- Concepts: capacity, latency, bandwidth, throughput
  - General / conceptual understanding is sufficient, no need exact values
- Storage hierarchy in a data center
- Capacity / latency / bandwidth over a data center (bottom right plot)
- Big Ideas (scale out, move processing to data, sequential processing, scalability)

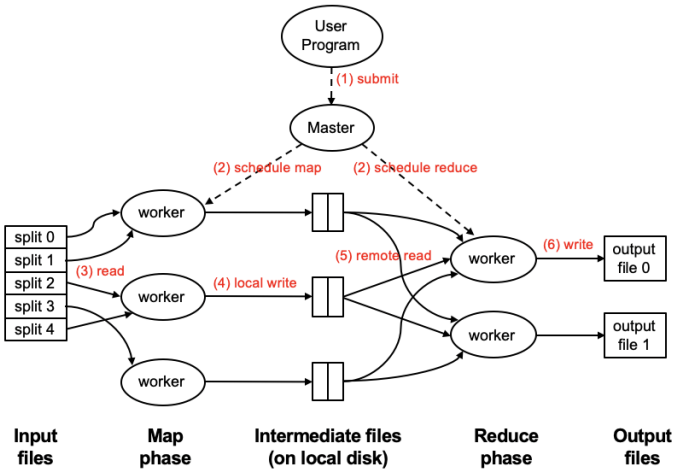




MapReduce

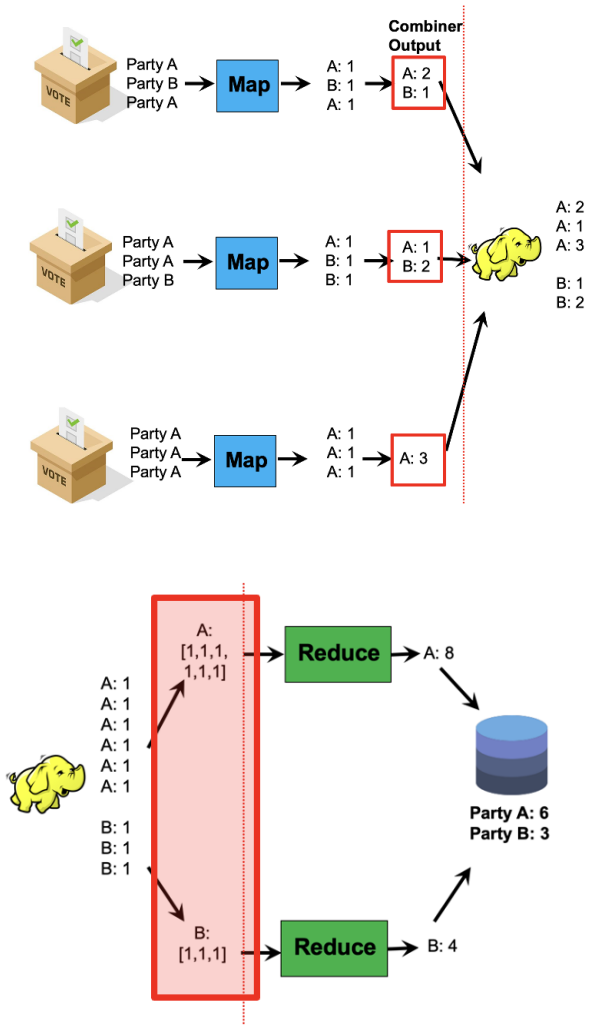
- Design principles: motivation, challenges, key ideas
- MapReduce
  - MapReduce interface
  - Ensuring correct types when writing MapReduce programs
  - Execution framework: responsibilities of Hadoop
- MapReduce framework imeplementation
  - Understand map, shuffle, reduce phases (no need for implementation details beyond the level discussed in lecture)
  - Understand how work is divided; difference between workers, tasks, functions, and tuples





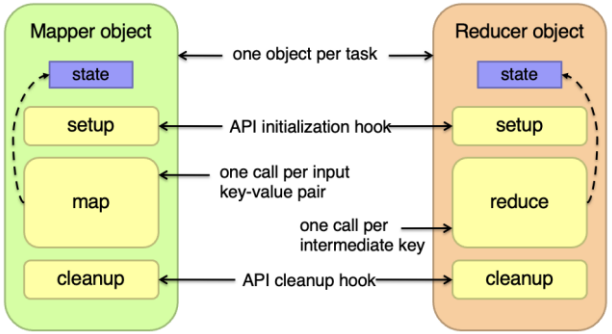
MapReduce

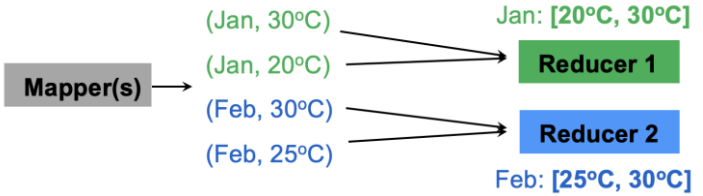
- Role of partitioner and combiner
  - Benefit of partitioners / combiners; correct usage of combiners
- Writing MapReduce programs
- Performance analysis: disk and network I/O, and working set memory usage



MapReduce

- Preserving State in MapReduce tasks: how to use it, what it can be used for
- [Optional] Secondary Sort / Composite Keys: how to use it, what it can be used for
- Regular combiners vs in-mapper combiners: usage, efficiency gains





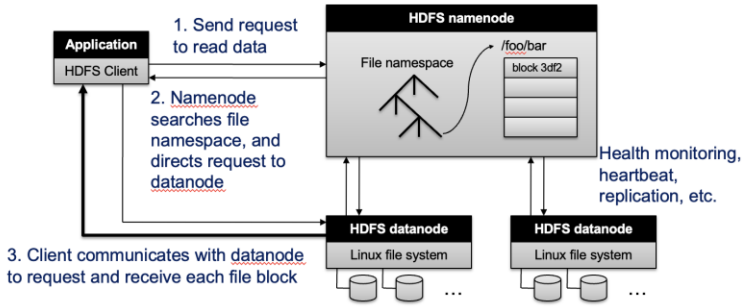

```
class Mapper {
  val counts = new Map()

  def map(key: Long, value: Text) = {
    for (word <- tokenize(value)) {
      counts(word) += 1
    }
  }

  def cleanup() = {
    for ((k, v) <- counts) {
      emit(k, v)
    }
  }
}
```

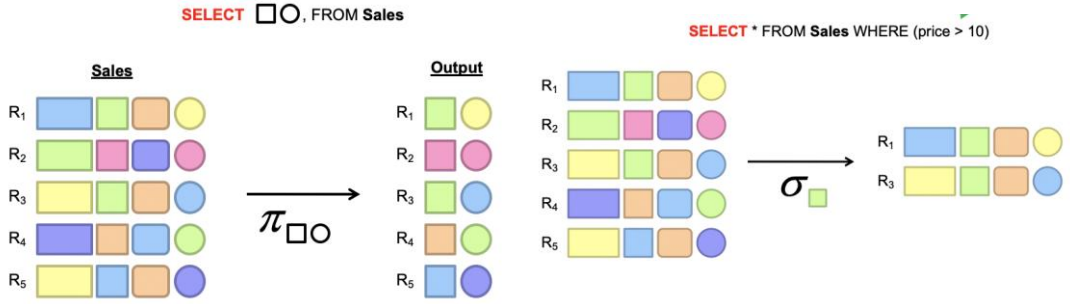
HDFS

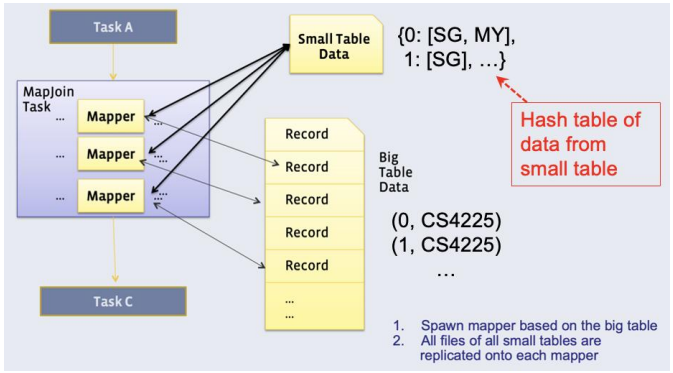
- Basic motivation and interface (no need to be familiar with specific commands)
- Basic infrastructure (namenode / datanode, reading / writing, replication)

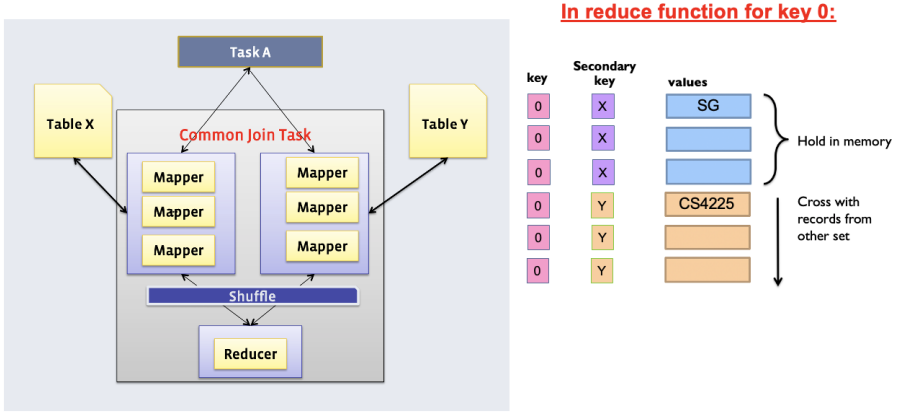


MapReduce and Relational Databases

- Understanding of algorithms and MapReduce implementation
  - Basic operations: Projection, Selection, Group By
  - Joins:
    - Broadcast Join
    - Reduce Side Join

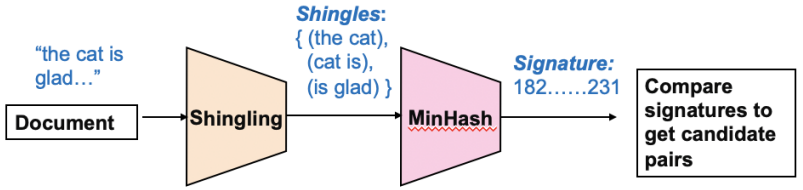


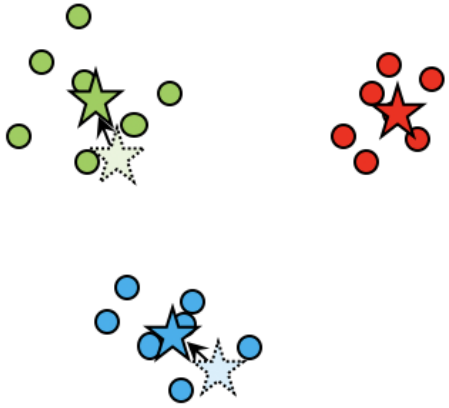




MapReduce and Data Mining

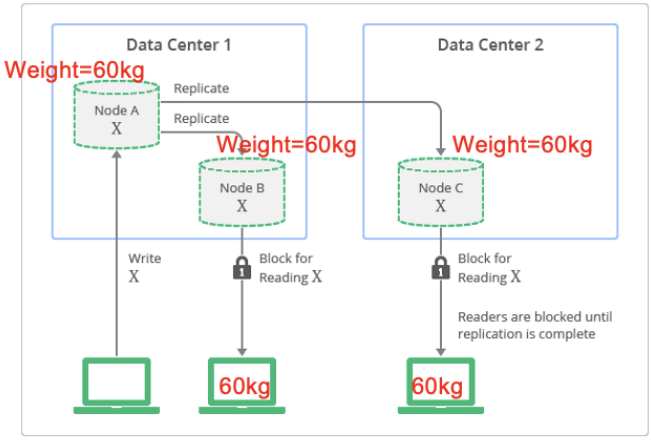
- Similarity Search
  - Similarity measures
  - Goals: similarity search vs all-pairs similarity
  - Shingling
  - MinHash algorithm & key property (no need for proof)
  - MapReduce implementation
- K-means clustering
  - K-means algorithm
  - MapReduce implementation
  - Efficiency analysis





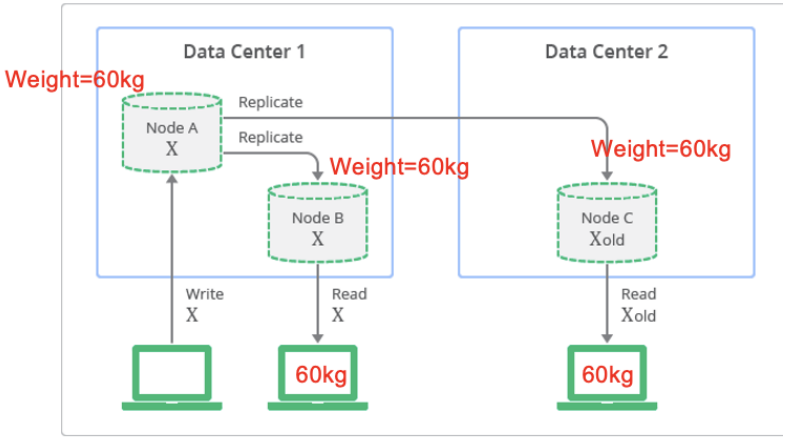
NoSQL

- Types of NoSQL databases
- Types of data suitable for key-value, document, graph, vector databases
- [Optional] Concepts: eventual consistency, duplication



All readers read new value (60kg)

Strong consistency: any reads immediately after an update must give the same result on all observers



Eventually, the update is propagated to Node C; after which, readers will read the correct value

Eventual consistency: if the system is functioning and we wait long enough, eventually all reads will return the last written value

Tips for Designing MapReduce Programs

- In exam, only pseudocode will be required (if you want to write in Python / Java, that is fine too). As long as your pseudocode is “reasonable” (understandable to the grader), it will be accepted
- Think about what key the mapper should emit: this key will be used to group the data for the reducers
- Then think about what value the mapper should emit: this should give the reducers all the information they need to perform their task
- Evaluating efficiency: main considerations are the disk and network I/O (determined by the amount of data emitted by mappers, but possibly reduced by combiners), and the memory working set (determined by the mapper’s intermediate state)

MapReduce: Example

Given two documents (documents 0 and 1), we want to find all k-shingles in document 0 but not in document 1 (that is, all shingles which appear at least once in document 0, but do not appear at all in document 1), where k=1. We receive our documents line by line, receiving input key-value pairs of the form `<docID, line>`, where `docID` (an integer of either 0 or 1) is the document ID being read, and line is a space-separated string containing a line of the document. For each k-shingle in the final answer, your reduce function should emit a tuple (only once) of the form `<shingle, 1>`. Show pseudo-code for how you would use MapReduce to find all such shingles. You can assume that the input (in line) is 'clean'; e.g. no duplicate spaces or spaces at the start / end of the line, and no characters other than letters and spaces are present. You can assume the existence of a string splitting function of your choice, e.g. `split()`.

Bonus: show how to use a combiner (not in-mapper combiner) to speed up the program.

```
map (docID, line) {
  /* your pseudo code*/
  /* output the map results by calling the API, emit(specify your map output)*/
}
reduce (/* specify your input to reducer */) {
  /* your pseudo code */
  /* output the map results by calling the API, emit(shingle, 1). */
}
```

MapReduce: Example

```
map (docID, line) {
  tokens = line.split()
  for token in tokens:
    emit(token, docID)
}

reduce (token, docIDs[]) {
  if 0 in docIDs and 1 not in docIDs:
    emit(token, 1)
}
```

Explanation:

- 1-shingles are just words (or tokens)
- The map() function should use words as keys, so that each reduce() will handle one word
- The map() needs to emit the docID as value, since the reduce() needs this information

MapReduce: Example

```
map (docID, line) {
  tokens = line.split()
  for token in tokens:
    emit(token, docID)
}

reduce (token, docIDs[]) {
  if 0 in docIDs and 1 not in docIDs:
    emit(token, 1)
}

combine (token, docIDs[]) {
  if 0 in docIDs:
    emit(token, 0)
  if 1 in docIDs:
    emit(token, 1)
}
```

Explanation:

- 1-shingles are just words (or tokens)
- The map() function should use words as keys, so that each reduce() will handle one word
- The map() needs to emit the docID as value, since the reduce() needs this information

Combiner:

- The combine() function combines multiple (token, 0) tuples into a single (token, 0) tuple, and similarly combines multiple (token, 1) tuples into a single (token, 1) tuple.
- Thus it does not change the final output, but speeds up the program by reducing the number of such tuples emitted (and thus the disk and network I/O).

Good luck with the exam!

Figures generated from DALL·E 3.

Acknowledgement and Others

- Some slides are adopted/revised from
  - Bryan Hooi
- Time management is the key.
  - Effective Time Management: Best Practices for Students and Educators. https://blog.nus.edu.sg/learners/2024/09/24/effective-time-management-best-practices-for-students-and-educators/.
  - 8 Time Management Tips for Students. https://summer.harvard.edu/blog/8-time-management-tips-for-students/.