MapReduce Emulator

CS2431 Lab3

MapReduce overview

- MapReduce: Simplified Data Processing on Large Clusters. Jeffrey Dean and Sanjay Ghemawat. OSDI 2004
- SIGOPS Hall of Fame Award 2015
- 15209 citations so far.
- Both authors won ACM SIGOPS Mark Weiser Award in 2012.

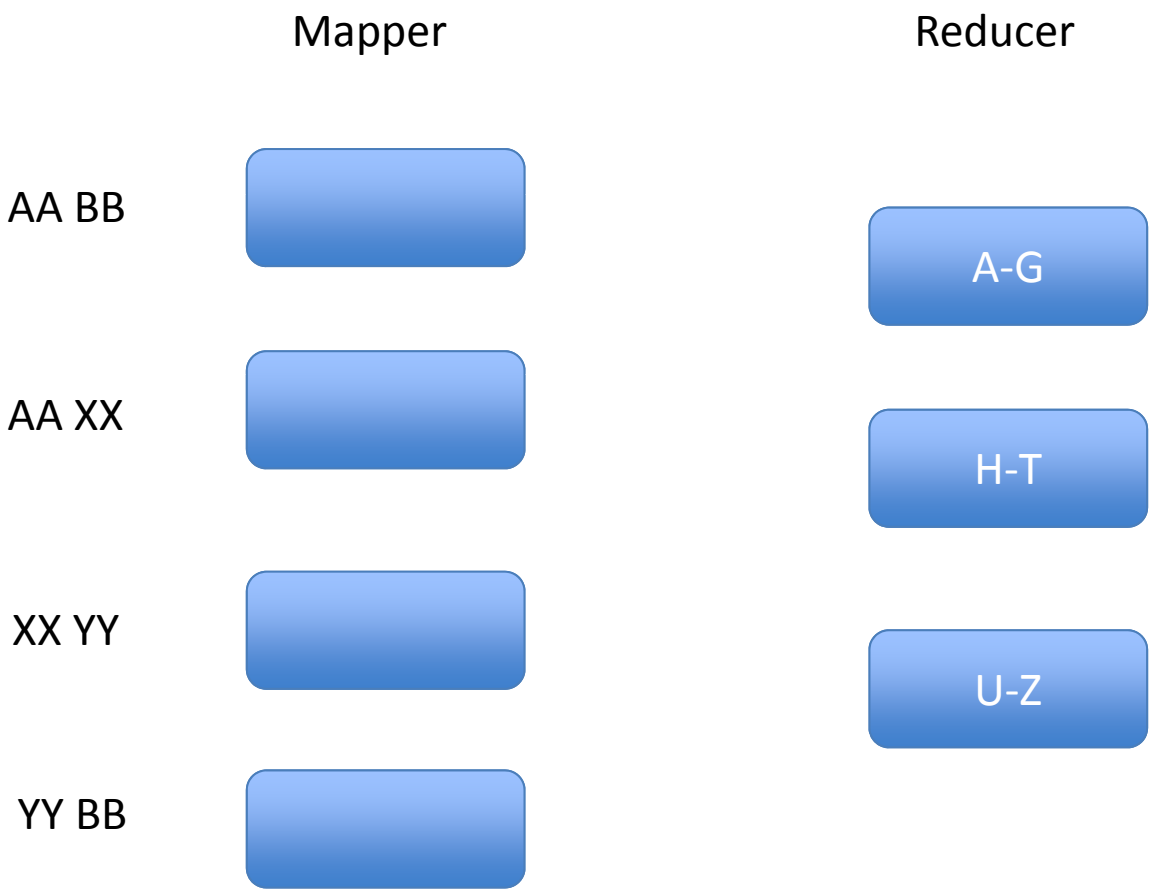
Lab3 overview

- Emulate the execution of “wordcount”, a typical MapReduce application
- Use the buffered_queue built in Lab 2
- Create producer and consumer threads

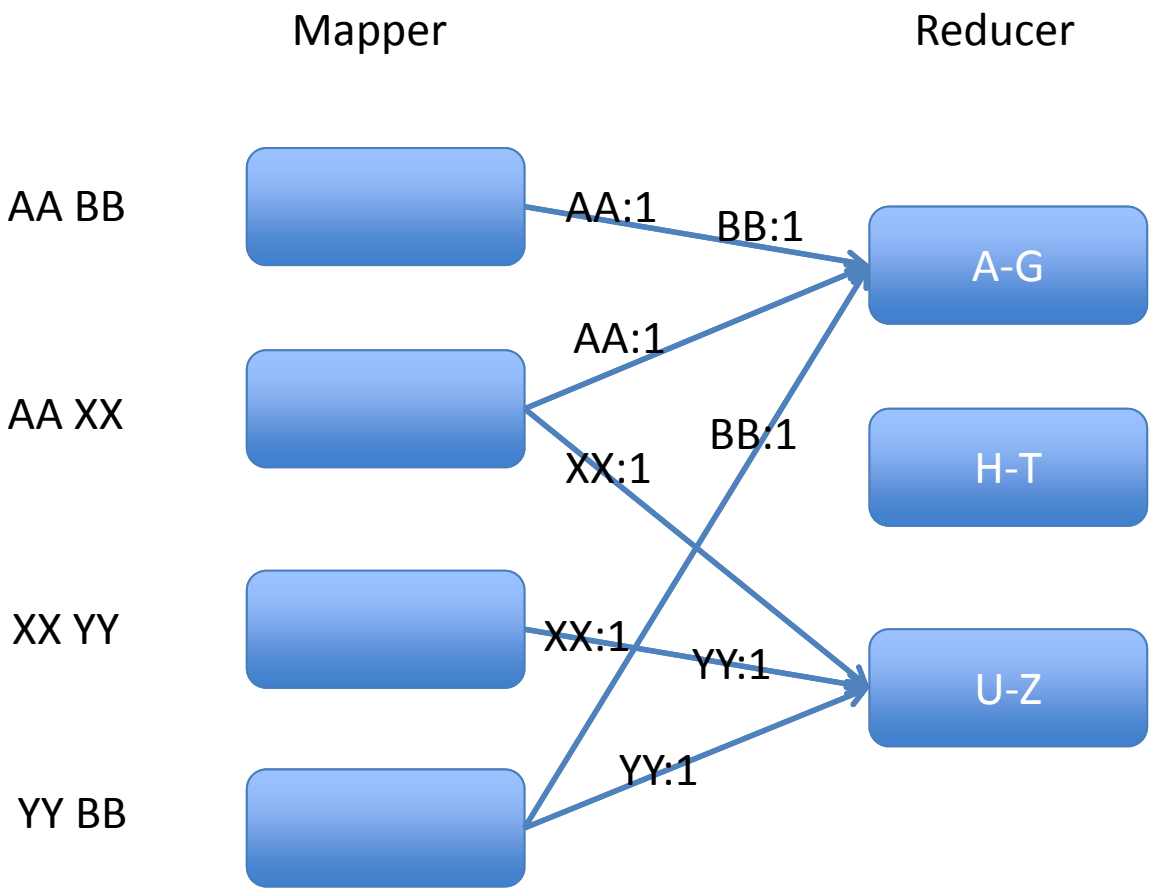
Wordcount

- Problem: give a certain document, count the number of occurrence of each word
- Example: For “AA BBB AAA AA BB BBB”, output “AA:2, BBB:2, AAA:1, BB:1”
- Simple?
- Not if you need to handle a big document
  - Google needs to do this for all webpages.
  - It cannot be handled by a single machine.

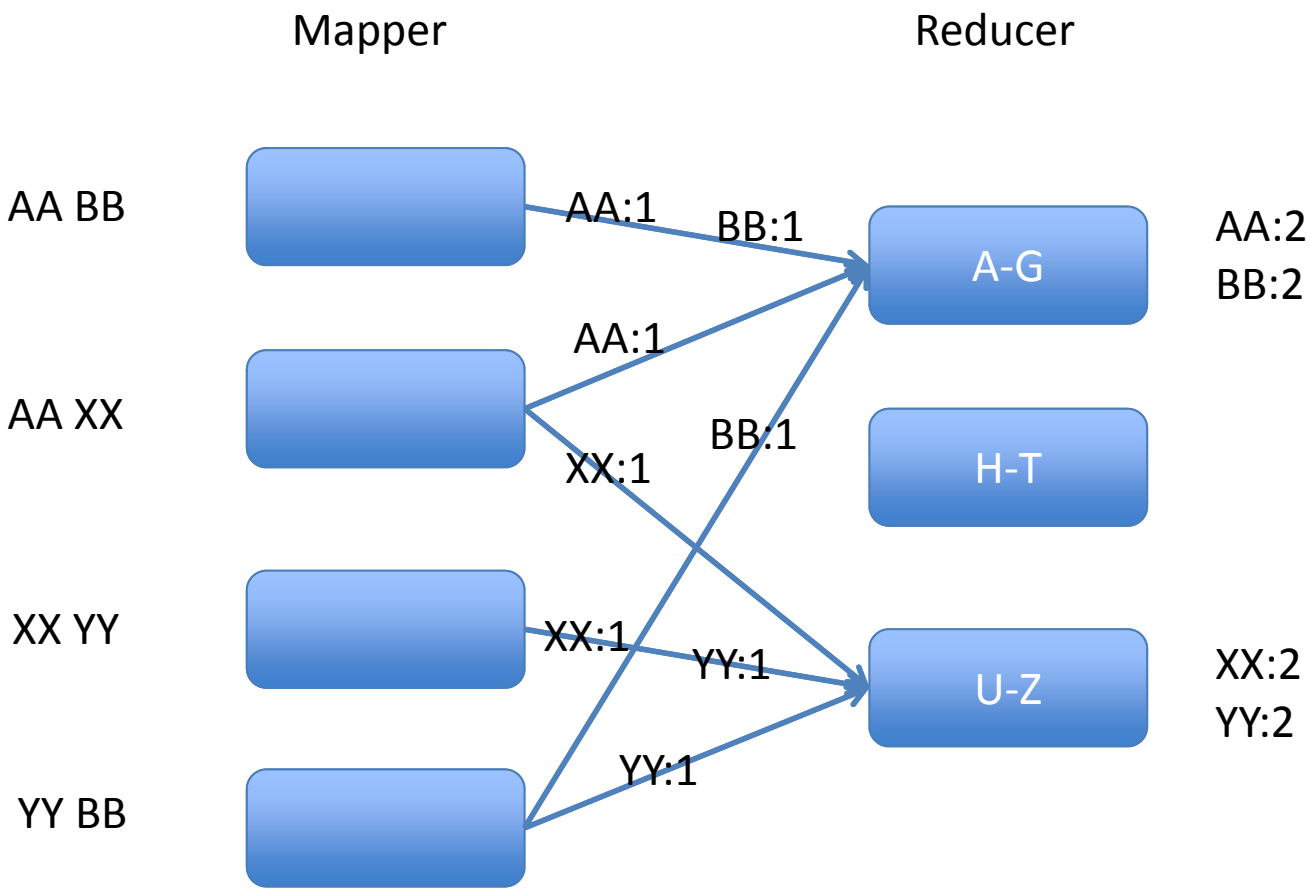
MapReduce wordcount



MapReduce wordcount



MapReduce wordcount



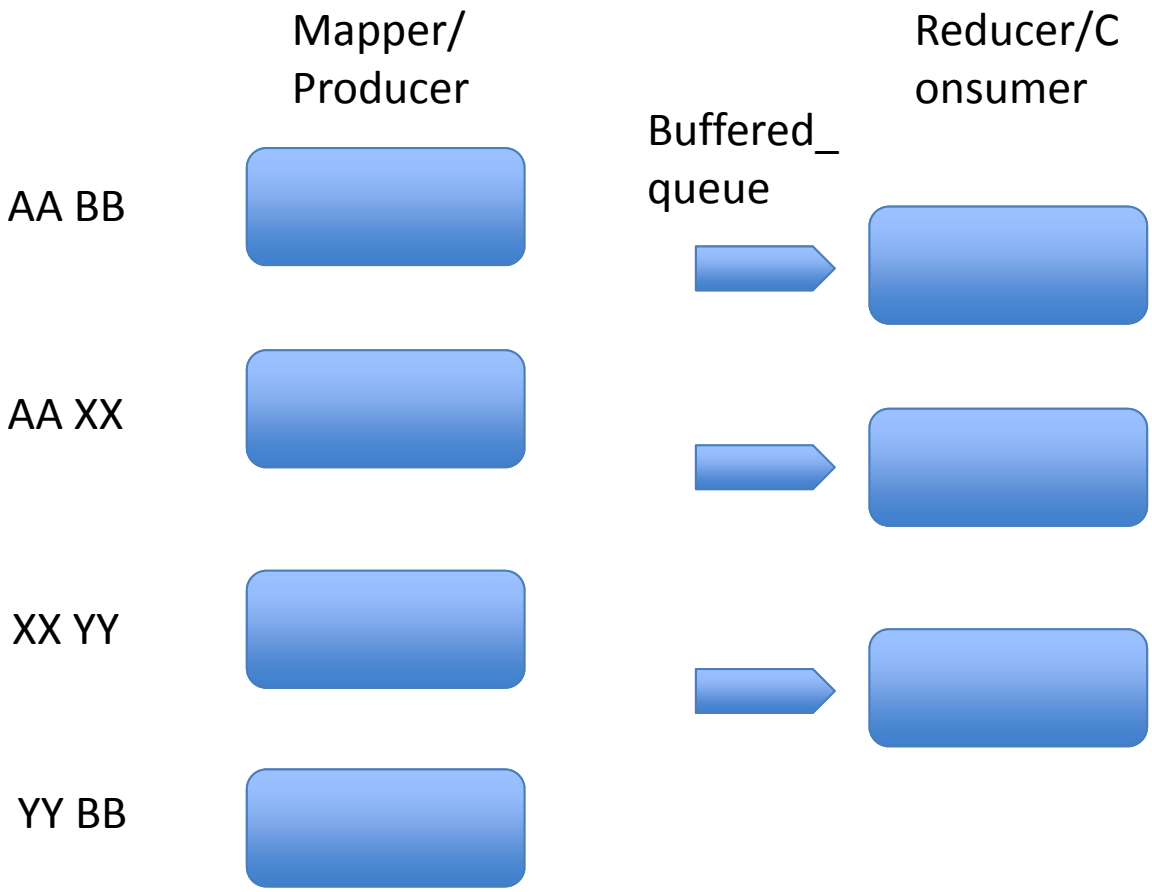
MapReduce wordcount

- Each mapper and reducer is a single process. They run on different machines.
- They communicate by using TCP/IP.
- Additional problems:
  - What to do if a machine crashes or becomes slow.
  - What to do if multiple users submit jobs.
  - ……

Lab3: A simplified MapReduce

- Each mapper and reducer is a thread. They run in the same process.
- They communicate by using buffered_queue.
  - Mapper is a producer; Reducer is a consumer.
- Ignore additional problems

Lab3: A simplified MapReduce



Lab3: Your job

- Create mapper threads.
  - Each mapper thread will need to process a string.
- Create reducer threads.
  - Each reducer threads is responsible for a buffered_queue.
- When a mapper thread parses a word, put it in the corresponding queue
- When a reducer thread gets a word from its queue, update its count.

Lab3: Additional questions

- How to distributed work around reducers:
  - One approach: reducer1 responsible for words starting with “A”, reducer2 for “B”, …
  - Not good for load balancing. Why?
  - Can you design a better solution?

- How can a reducer know the work is done?

Lab3: Your job

- void wordcount(int m, int r, char** docs)

- m: number of mapper threads
- n: number of reducer threads
- docs: m strings, one for each mapper

- A string is composed of only “a”-“z” and space.
- Finally you should print the count for each word.

Questions?