Big Data Technologies and Datasets

Lecture 5 – Map Reduce

Based on slides from multiple sources

Outline

- What is Map Reduce?
- Map Reduce examples
- Parallel execution
- Summary

Map Reduce

חוסך את כאב הראש שנוצר כאשר אני רוצה לכתוב קוד שאמור לרוץ בכמה מחשבים שונים, ומאפשר לי להתרכז בכתיבת הקוד שאני צריך ספציפית

[Dean et al., OSDI 2004, CACM Jan 2008, CACM Jan 2010]

- Overview:
  - Simple & powerful programming paradigm for large-scale data analysis
  - An associated parallel and distributed system for commodity clusters
- Pioneered by Google
  - Processes 20 PB of data per day
- Popularized by open-source Hadoop project
  - Used by Yahoo!, Facebook, Amazon, and the list is growing

3

Typical flow of MapReduce

את שתי הפונקציות הללו, פונקציית מאפ ופונקציית רדיוס, אני כותב - נרחיב בהמשך

- Read a lot of data
- Map: extract something you care about from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, or transform
- Write the results

- Outline stays the same!
- Map and Reduce change accordingly…

4

Map-Reduce Algorithms

- Map tasks convert inputs to key-value pairs.
  - “keys” are not necessarily unique.
- Outputs of Map tasks are sorted by key, and each key is assigned to one Reduce task.
- Reduce tasks combine values associated with a key.

Formal Definition

- Programmer specifies two primary methods:
  - Map: $<k_1, v_1>$: list($<k_2, v_2>$)

    פעולת המאפ מקבלת כאינפוט מפתח וערך, ומוציאה רשימה של זוגות של מפתח וערך
  - Reduce: $<k_2$, list(v_2)$>$: list($<k_3, v_3>$)

    פעולת הרדיוס מקבלת מפתח ורשימת ערכים לכל מפתח, ומוציא רשימה של מפתחות אחרים וערכים אחרים
  - All v_2 with the same k_2 are reduced together.
  - (Remember the invisible “Shuffle and Sort” step)

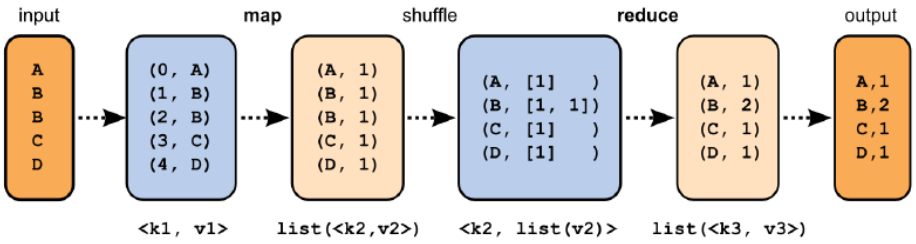
Simple data flow example

נניח שכל מה שאני רוצה לעשות זה לקחת קובץ גדול, ולספור שכיחות מילים

פעולת השאפל: מחזירה כמה פעמים הופיעה כל אות. מכינה את האינפוט לפעולת הרדיוס

פעולת הרדיוס: מבצעת את האגרגציה

- Counting letters



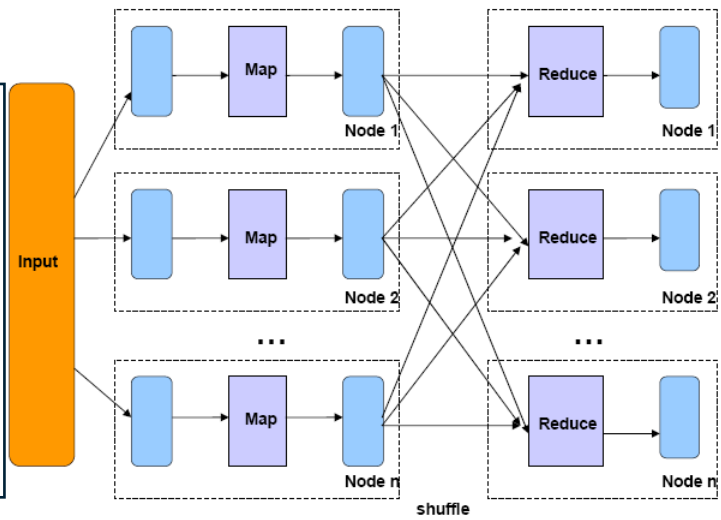
.פעולות המאפ והרדיוס יכולות לרוץ על גבי מחשבים שונים

אני (למעשה המתכנת) הוא זה שכותב את פונקציות המאפ והרדיוס. **חובה חובה חובה!!!** לכתוב מהו המפתח והערך שנכנס לפונקציית המאפ - כלומר חובה לרשום מה פעולת המאפ מקבלת - איזה גודל של גוש (תו, אלף תווים, שורה). כלומר, חושב לכתוב מה הוא מקבל. לגבי הרדיוס, לא חשוב לרשום מה הוא מקבל, מכיוון שזה דיי ברור מפונקציית המאפ. .בכל זאת, מומלץ כן לכתוב מה פונקציית הרדיוס מקבלת

7

Data flow example with parallelism

המערכת מחלקת את האינפוט לגושים, ושולחת כל גוש ליחידת מחשוב אחרת . פעולת הרדיוס לוקחת מפתח מסוים מכל שרת ועובדת עליהם .בצורה מרוכזת
כמה פעולות מאפ אני ארצה שיהיו? אם יש לי 10 מחשבים, ארצה שיהיו לי יותר, נניח 40 פעולות מאפ. כך אני מנצל יותר את המחשבים כך שכולם יהיו עסוקים כל הזמן (במידה ויש לי מחשבים שהם יותר מהירים ואיטיים, למשל). אבל, לא ארצה כמות עצומה של פעולות מאפ



This is a **simplistic** View: Later on we will discuss how this parallelism **really** works

8

Map Reduce examples

Example 1: Word Frequencies in Web Pages

Input: files with one document per record

Specify a *map* function that takes a key/value pair
key = document URL
value = document contents

Output of map function is (potentially many) key/value pairs.
In our case, output (word, "1") once per word in the document

"document1", "to be or not to be"

↓

"to", "1"
"be", "1"
"or", "1"

Example 1: Word Frequencies in Web Pages (contd.)

MapReduce library gathers together all pairs with the same key (shuffle/sort)

Specify a *reduce* function that combines the values for a key
In our case, compute the sum

key = "be" values = "1", "1"	key = "not" values = "1"	key = "or" values = "1"	key = "to" values = "1", "1"
↓	↓	↓	↓
"2"	"1"	"1"	"2"

Output of reduce (usually 0 or 1 value) paired with key and saved

"be", "2"
"not", "1"
"or", "1"
"to", "2"

Example 1: Word Frequencies in Web Pages (contd.)

Pseudo Code:

```
map(key, value):
// key: document name; value: text of document
    for each word w in value:
        emit(w, 1)

reduce(key, values):
// key: a word; values: list of 1s (a single 1 for each such word that a appeared in a map
        result = 0
        for each v in values:
            result += v
        emit(key, result)
```

Combiners

הפעלת הרדיוס כמה פעמים

- Often a map task will produce many pairs of the form (k_2,v_2), (k_2,v_2'), (k_2,v_2''), … for the same key k_2.
  - E.g., popular words in the Word Count example
- Can save network time by pre-aggregating values at the mapper: combine(<k_2, list(v_2)>) → list(<k_3,v_3>)
- Combiner is usually same as the reduce function
- Works only if reduce function is commutative and associative

commutative: A+B == B+A
associative: A + (B+C) == (A+B) +C

13

Example 2: Join

- Suppose we want to compute R(A,B) JOIN S(B,C)
  - I.e., find tuples with matching B-values.
- *R* and *S* are each stored in a chunked file.

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

⋈

B	C
b_2	c_1
b_2	c_2
b_3	c_3

=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

14

Example 2: Join (contd.)

- Map (receives a single tuple as input):
  - Tuple R(a,b) to <b, R(a,b)>
  - Tuple S(b,c) to <b, S(b,c)>

- Reduce (receives a set of tuples from either R or S that have the same b value) :
  - For each record r(a, b) in this set originated from R:
    - For each record s(b, c) in this set originated from S:
      Emit (a, b, c)

15 15

Question

- What if we want to limit ourselves to using *k* Reduce tasks at max?

מה אם הייתי רוצה להגביל את כמות ה"רדיוס" שאני עושה?
הייתי עושה מודולו למפתחות שלי, נניח מודולו 10, ואז הייתי יכול לעשות רק עשר פעולות רדיוס. דוגמה לכך בקובץ האקסל, בגיליון השני.

16

Example 3: Join (contd.)

- Use a hash function *h* from B-values to *k* buckets.
  - Bucket = Reduce task.
- Map (receives a single tuple as input):
  - Tuple R(a,b) to <h(b), R(a,b)>
  - Tuple S(b,c) to <h(b), S(b,c)>

Example 3: Join (contd.)

- Reduce (receives a set of tuples from either R or S that have the same h(b) value) :
  - For each record r(a, b1) in this set originated from R:
    - For each record s(b2, c) in this set originated from S:

```
If b1==b2:
    Emit (a, b, c)
```

Example 3: Join (contd.)

- Key point: If R(a,b) joins with S(b,c), then both tuples are sent to Reduce task h(b).
- Thus, their join (a,b,c) will be produced there and shipped to the output file.

19

Exercise

מתורגל גם בתרגיל הבית הבא

- Consider a very large csv file that contains the following fields:
  - Timestamp, Product ID, Price Paid, Customer ID
- Write pseudo-code for a Map-Reduce job that will calculate the total amount that each customer paid in each month.

פתרון:
עבור פעולת המאפ - המפתח יהיה מזהה הלקוח, והערך יהיה רשימה של המחיר ששולם והטיימסטמפ (כל המידע שאני צריך מהרשומה כדי לענות על הדרישה). זאת כדי שאוכל לסכום על כל המחירים ששולמו.
עבור פעולת הרדיוס - נפריד בין החודשים השונים של הלקוח, ועבור כל חודש לסכום את כמה שהוא הוציא.

אם כמות הקניות שלקוח עושה היא עצומה, ואני לא יכול להתמודד עם פעולת רדיוס אחת בלבד, מה נעשה?
עבור פעולת המאפ - נגדיר את המפתח שיהיה מזהה הלקוח ואת החודש (שאני יכול להוציא מהטיימסטמפ). הערך יהיה המחיר ששולם.
עבור הרדיוס - כל רדיוס יקבל לקוח וחודש, וכמובן את כל המחירים שהוא שילם בכל החודש, ואז לסכום על כולם ולהחזיר את מזהה הלקוח והסכום הכולל כך למעשה אפשרנו להגדיל את הביזור.

20

Parallel execution

נרחיב כעת על הרקע, מה קורה בעת הביצוע

21

Master and Worker

עבור מערכת שמריצה מאפ רדיוס - Hadoop לדוגמה

- One master, many workers
  - Input data split into M map tasks (typically 64 MB in size)
  - 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 task
  - Worker reads task input (often from local disk!)
  - Worker produces R local files containing intermediate (k,v) pairs

המאסטר משייך את המשימות לעובדים

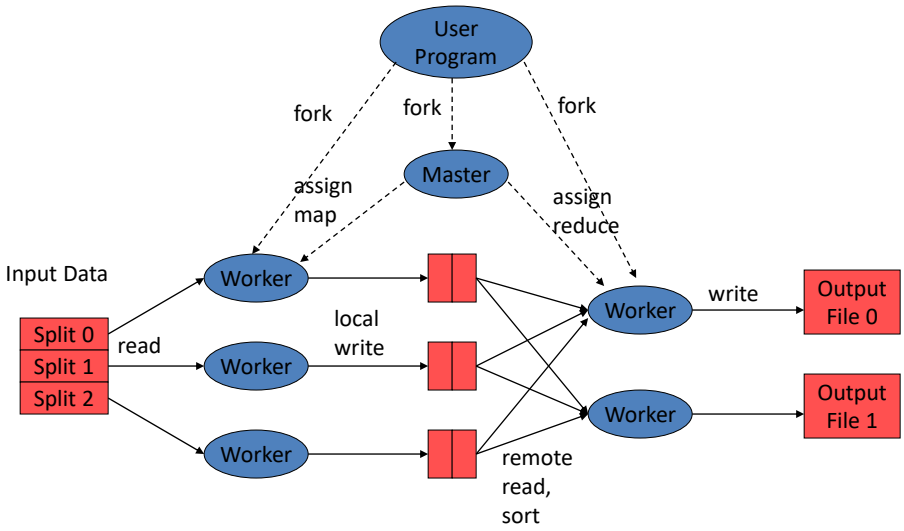
- Master assigns each reduce task to a free worker
  - Worker reads intermediate (k,v) pairs from (all) map workers
  - Worker sorts & applies user's Reduce op to produce the output

לאחר שהמאסטר מסיים את כל פעולות המאפ, הוא ניגש לבצע את כל פעולות הרדיוס

22

תיאור סכמתי של השקף הקודם:

Distributed Execution Overview



Data flow

- Input and final output are stored on a distributed file system
  - Scheduler tries to schedule map tasks "close" to physical storage location of input data.

המאסטר מנסה לנתב את העבודה כך שתתבצע מינימום העברה של מידע ממחשב למחשב

- Intermediate results are stored on local FS of map and reduce workers
- Output is often input to another map reduce task

לפעמים נמצא את עצמנו משרשרים אוסף של פעולות מאפ-רדיוס, אחת אחרי השניה.

Coordination

- Master data structures
  - Task status: (idle, in-progress, completed)
  - Idle tasks get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its R intermediate files, one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to detect failures

25

Responsibility of the Master

- Assign Map and Reduce tasks to Workers.
- Check that no Worker has died.
- Communicate results of Map to the Reduce tasks.

26

Communication from Map to Reduce

דילג

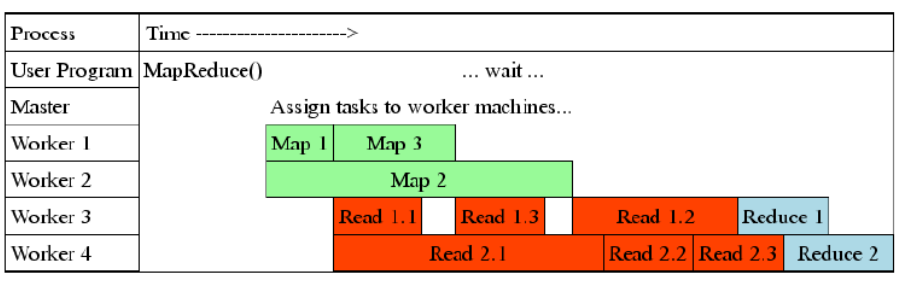
- Select a number R of reduce tasks.
- Divide the intermediate keys into R groups, e.g. by hashing.
- Each Map task creates, at its local FS, R files of intermediate key-value pairs, one for each Reduce task.

27

Task Granularity

סכמה של משימה שיש לעשות
עבור מאסטר וארבעה עובדים

- Fine granularity tasks = many more map tasks than machines
- Improves dynamic load balancing
- Speeds up recovery from worker failure
- Allows the shuffling process to be parallelized with map execution



רק אחרי שסיימתי לקרוא את כל המאפרים אני יכול
(להתחיל לבצע את הרדיוס (לא משנה עבור איזו משימה

28

Typical sizes of M and R

M == Map, R == Reduce

- Rule of a thumb:
  - Make M and R much larger than the number of nodes in cluster
  - One DFS chunk per map is common
- Usually R is smaller than M because output is spread across R files
  - Example in Google: M=200,000; R=4,000; workers=2,000

29

Fault Tolerance via Re-Execution

- Failure modes:
  - Compute node failure (e.g., disk crash).
  - Rack communication failure.
  - Software failures, e.g., a task requires Java n; node has Java n-1.
- Addressing Worker Failures
  - Detect failure via periodic heartbeats
  - Re-execute completed and in-progress map tasks
  - Re-execute in-progress reduce tasks
  - Task completion committed through master
- Addressing Master failure:
  - State is checkpointed to replicated file system
  - New master recovers & continues
- Very Robust: lost 1600 of 1800 machines once, but finished fine

30

Refinement: Redundant Execution

- Slow workers significantly delay completion time
  - Other jobs consuming resources on machine
  - Bad disks w/ soft errors transfer data slowly
  - Weird things: processor caches disabled (!!)
- Solution: Near end of phase, spawn backup tasks
  - Whichever one finishes first "wins"
- Dramatically shortens job completion time

31

Refinement: Locality Optimization

שורה תחתונה: אני ארצה שכל מחשב יריץ פעולה על מידע שכבר נמצא באותו המחשב. כך נמנעת העברת מידע, וננצל כמה שיותר את הלוקאליטי

- Master scheduling policy:
  - Asks GFS for locations of replicas of input file blocks
  - Map tasks typically split into 64MB (GFS block size)
  - Map tasks scheduled so GFS input block replica are on same machine or same rack
- Effect
  - Thousands of machines read input at local disk speed
    - Without this, rack switches limit read rate

32

שקף לא מעודכן...

MapReduce Experience

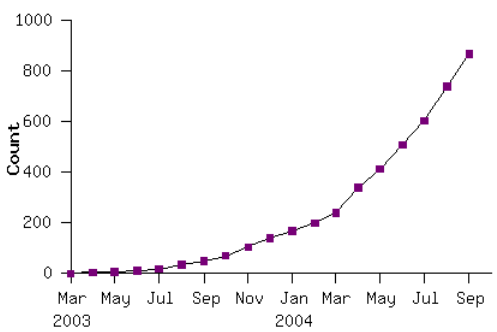
- Runs on large commodity clusters:
  - 1000s to 10,000s of machines
- Processes many terabytes of data
- Easy to use since run-time complexity hidden from the users
- 1000s of MR jobs/day at Google (circa 2004)
- 100s of MR programs implemented (circa 2004)

33

שקף לא מעודכן...

Model is Widely Applicable

MapReduce Programs In Google Source Tree



Example uses:

distributed grep	distributed sort	web link-graph reversal
term-vector / host	web access log stats	inverted index construction
document clustering	machine learning	statistical machine translation
...	...	...

34

Summary

35

MapReduce Advantages

אנו מתמקדים רק בלוגיקה של הקוד,
ולא במעטפת של המקבול והכל...

- Automatic Parallelization:
  - Depending on the size of RAW INPUT DATA ➔ instantiate multiple MAP tasks
  - Similarly, depending upon the number of intermediate <key, value> partitions ➔ instantiate multiple REDUCE tasks
- Run-time system handles:
  - Data partitioning
  - Task scheduling
  - Inter-machine communication
  - Machine failures
- Completely transparent to the programmer/analyst/user

36

Takeaway

- MapReduce's data-parallel programming model hides complexity of distribution and fault tolerance
- Principal philosophies:
  - Make it scale, so you can throw hardware at problems
  - Make it cheap, saving hardware, programmer and administration costs (but requiring fault tolerance)
- Hive and Pig further simplify programming
- MapReduce is not suitable for all problems, but when it works, it may save you a lot of time

37

MapReduce: A step backwards?

חסרונות של מאפ-רדיוס

- Don't need 1000 nodes to process petabytes:
  - Parallel DBs do it in fewer than 100 nodes
- No support for schema:
  - Sharing across multiple MR programs difficult
- No indexing:
  - Wasteful access to unnecessary data
- Non-declarative programming model:
  - Requires highly-skilled programmers
- No support for JOINs:
  - Requires multiple MR phases for the analysis

38

Map Reduce vs Parallel DBMS

[Pavlo et al., SIGMOD 2009, Stonebraker et al., CACM 2010, …]

	Parallel DBMS	MapReduce
Schema Support	✓	Not out of the box
Indexing	✓	Not out of the box
Programming Model	Declarative (SQL)	Imperative (C/C++, Java, …) **Extensions through Pig and Hive**
Optimizations (Compression, Query Optimization)	✓	Not out of the box
Flexibility	Not out of the box	✓
Fault Tolerance	Coarse grained techniques	✓
Number of Nodes	A few large nodes	10k of commodity servers

39