1) Bloom Filters (και πως μπορουμε να μειωσουμε πιθανοτητα λαθους – με αυξηση του πίνακα):

2) Hash / Broadcast join

3) Sort-Merge Join Algorithm

4) Map reduce on Very Large vs Very Small Datasets

5) Lazy execution (τι είναι και πλεονεκτήματα-οφέλη)

6) Reduce Side Join (τι είναι, pros cons) σύγκριση με Map side Join

7) Map Side Join δες kai 6)

8) SQL optimization ή query optimization

9) Narrow-Wide Dependencies

10) Columnal vs Row databases (query use cases)

11) Shuffle & Sort in MapReduce

12) Πώς γίνεται προσπέλαση σε εγγραφές σταθερού μήκους;

13) Πως γίνεται συνένωση αν οι πίνακες είναι ταξινομημένοι; - Δες το 3)

14) Τι συμβάλλει στο κόστος ενός sql ερωτήματος;

15) Map Reduce κοντά στα δεδομένα (data locality)

16) Τι είναι η partition function του MapReduce και ποια είναι η default στο Spark?

17) Τι συμβαίνει όταν αποτυγχάνει ένας Mapper, ενώ στέλνει τα δεδομένα σε έναν Reducer και κόβεται στη μέση η όλη διαδικασία αποστολής;

18) Πώς δουλεύει το MapReduce;

19) HDFS γιατί έχει υψηλή διαθεσιμότητα;

20) Πώς εξασφαλίζεται το fault tolerance se HDFS?

21) Ποιοί παράγοντες μας νοιάζουν στην βελτιστοποίηση;

22) Πώς γίνεται distributed join με ένα μικρό και ένα μεγάλο dataset? Δες Hash και Broadcast 2) ???Ή ΟΧΙ?

23) Διαφορές GFS – HDFS

24) Τι είναι οι Combiners?

25) Map vs Reduce side δες 6)

26) Spark Hash vs Sort shuffle

1) Bloom Filters (και πως μπορουμε να μειωσουμε πιθανοτητα λαθους – με αυξηση του πίνακα):

Είναι βελτιστοποίηση του Big Table

Έξυπνο ευρετήριο που χρησιμοποιείται ως ένας χάρτης από bit που εσύ διαλέγεις πόσο μεγάλο θα το κάνεις, το οποίο συμβουλέυεσαι πριν αναζητήσεις ένα κλειδί σε μια αποθηκευτική δομή. Άρα έχει παραπάνω αποθηκευτικό χώρο που φορτώνεται στη μνήμη, ώστε να συμβουλευτείς το bloom filter αν υπάρχει το κλειδί που θες ή όχι στην δομή.

Αν σου πει όχι γλυτώνεις άσκοπες αναζητήσεις και είναι guarantee ότι δεν είναι εκεί γιατί δεν έχεις ποτέ false negatives.

Μπορεί να πάρεις false positives, αλλά γλυτώνεις από τα true negatives.

Ο χρήστης καθορίζει αν θα χρησιμοποιηθούν

Χρησιμοποιείται για να καθοριστεί αν κάποιο SSTable περιέχει δεδομένα από συγκεκριμένο row χωρίς να το ανακτήσουμε

Με λίγο αποθηκευτικό χώρο παραπάνω μπορούμε να αποκλείουμε αμέσως μεγάλο αριθμό sstables

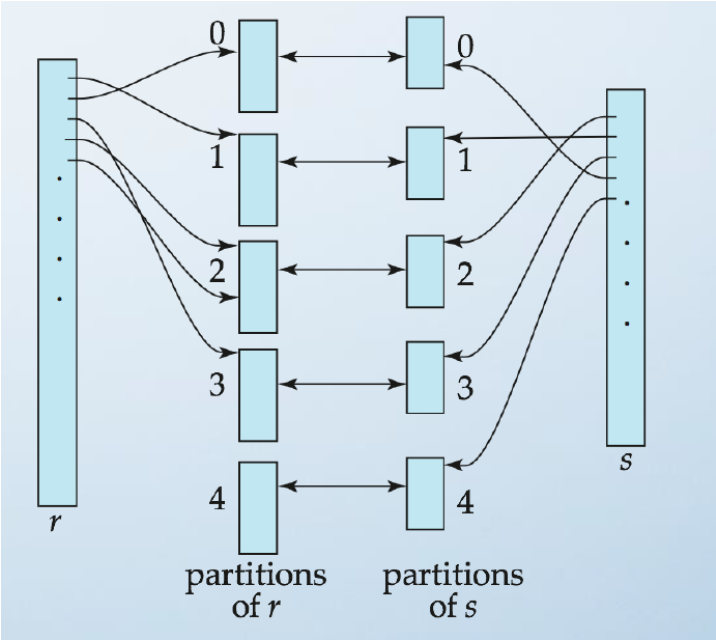
Πιθανά false positives: Σε αυτή την περίπτωση απλά δεν θα βρει κάτι στο sstable

Ποτέ false negatives: ότι δεν βρίσκει, σίγουρα δεν υπάρχει.

Δημιουργούνται βάση locality group

2) Hash / Broadcast join

Περιγραφή Hash Join:

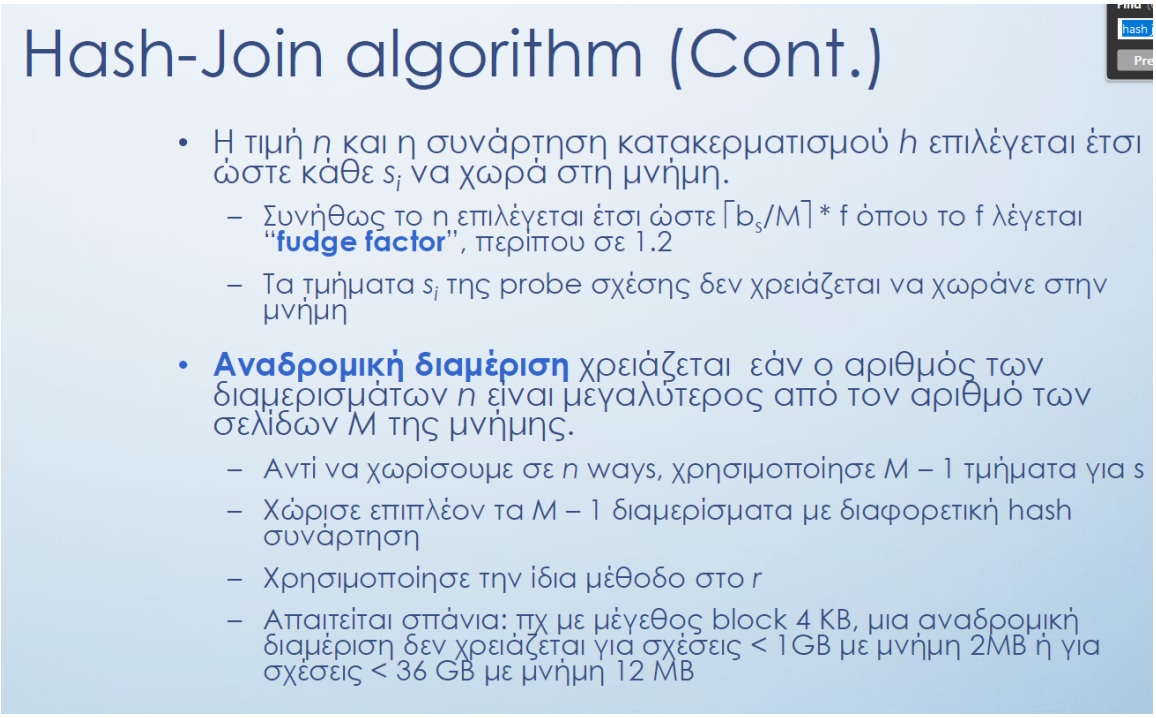


- Οι πλειάδες r_i *της σχέσης r χρειάζεται να συγκριθούν μόνο με τις πλειάδες* s_i *της σχέσης s*
- Δεν χρειάζεται να συγκριθούν με πλειάδες της s σε κανένα άλλο τμήμα, καθώς:
  - Μια πλειάδα r και μια πλειάδα s που ικανοποιούν την συνθήκη συνένωσης θα έχουν την ίδια τιμή για τα πεδία συνένωσης
  - Εάν η συνάρτηση κατακερματισμού έχει υπολογίσει μια τιμή i (που την ανέθεσε στον κάδο i) η πλειάδα r πρέπει να είναι στον κάδο r_i και η πλειάδα *s* στον κάδο s_i

Η συνένωση κατακερματισμού των *r* και *s υπολογίζεται ως εξής*.

1. Τεμάχισε την σχέση s χρησιμοποιώντας την συνάρτηση κατακερματισμού h. Κατά την κατάτμηση της σχέσης, ένα μπλοκ μνήμης δεσμεύεται ως το buffer εξόδου για κάθε διαμέρισμα.
2. Τεμάχισε την *r* παρομοίως.
3. Για κάθε *i:*
   1. Φόρτωσε το s_i στη μνήμη και δημιούργησε ένα ευρετήριο κατακερματισμού στη μνήμη χρησιμοποιώντας το πεδίο συνένωσης. Αυτό το ευρετήριο κατακερματισμού χρησιμοποιεί διαφορετική συνάρτηση κατακερματισμού από την h (αλλιώς θα πάνε πάλι όλα τα πεδία σε ένα τμήμα).
   2. Διάβασε τις πλειάδες στην r_i από το δίσκο μια προς μια. Για κάθε πλειάδα t_r εντόπισε κάθε αντίστοιχη πλειάδα t_s σε s_i χρησιμοποιώντας το ευρετήριο κατακερματισμού στη μνήμη. Βγάλε τη συνένωση των χαρακτηριστικών τους.

Η σχέση *s* λέγεται το **build input** και η σχέση *r* λέγεται το **probe input.**



Βασική ιδέα

Φόρτωση ενός συνόλου δεδομένων στη μνήμη σε ένα πίνακα κατακερματισμού (hashmap), με κλειδί το κλειδί συνένωσης

Ανάγνωση του άλλου συνόλου δεδομένων, διερεύνηση (probe) για κλειδί συνένωσης

Λειτουργεί εάν

R << S και το R χωράει στην μνήμη RAM

Υλοποίηση MapReduce :

Διανομή του R σε όλους τους κόμβους πχ μέσω της DistributedCache

Map στο S, κάθε mapper φορτώνει το R στη μνήμη και δημιουργεί το hashmap

Για κάθε πλειάδα στο S , έλεγχος (probe) του κλειδιού συνένωσης στο R

Δεν απαιτούνται reducers εκτός αν θέλουμε να κάνουμε κάτι άλλο

Περιγραφή Broadcast Join:

Συνένωση Μετάδοσης (broadcast join) (Alg. A4)

Στις περισσότερες εφαρμογές , |R| << |L|

Αντί να μετακινούμε τόσο το R όσο και το L στο δίκτυο

Μεταδίδουμε τον μικρότερο πίνακα R για να αποφύγουμε την επιβάρυνση δικτύου

Μια δουλειά map only (για HIVE) ή στενής εξάρτησης (narrow dependency για το Spark)

Κάθε εργασία map χρησιμοποιεί έναν πίνακα κατακερματισμού κύριας μνήμης για L ή R

3) Sort-Merge Join Algorithm

- Η ταξινόμηση των δεδομένων παίζει σημαντικό ρόλο για την επεξεργασία ερωτημάτων
  - τα ερωτήματα SQL μπορούν να καθορίσουν ότι η έξοδος θα ταξινομηθεί
  - πολλές από τις σχεσιακές λειτουργίες, όπως **οι συνενώσεις**, μπορούν να εφαρμοστούν αποτελεσματικά εάν οι σχέσεις εισαγωγής ταξινομηθούν πρώτα
- Η ταξινόμηση μιας σχέσης με την δημιουργία ενός ευρετηρίου (non-clustered) είναι **λογική**: στην πράξη τα δεδομένα δεν είναι ταξινομημένα: μόνο οι δείκτες του ευρετηρίου είναι.
  - Αυτό σημαίνει ότι μια σάρωση του ευρετηρίου για ανάκτηση κλειδιών μπορεί να σημαίνει πολλαπλές αναζητήσεις/μεταφορές, **μια για κάθε κλειδί**
- Αλγόριθμος external sort merge: ταξινόμηση για σχέσεις που δεν χωράνε όλες στην μνήμη (πιο γενικός)

- Στο πρώτο στάδιο, ένας αριθμός από ταξινομημένα **τρεξίματα (runs)** δημιουργείται (με υποσύνολα των εγγραφών. Σε δεύτερο στάδιο συγχωνεύονται (merge). Έστω ότι ο buffer χωράει M blocks

- Εάν βγουν πιο πολλά από M runs, και δεν χωράνε όλα στην μνήμη:
  - Δεν μπορώ να δώσω 1 block για κάθε merge
- Η συγχώνευση προχωράει σε πολλαπλά περάσματα, με M-1 runs εισόδου (δεν χωράει στον buffer όλα)
- Αρχικό πέρασμα: Συγχωνεύει τα πρώτα M – 1 τρεξίματα για να πάρει μόνο ένα τρέξιμο για το επόμενο πέρασμα. Στη συνέχεια, συγχωνεύει τα επόμενα M - 1 τρεξίματα, και ούτω καθεξής, έως ότου έχει επεξεργαστεί όλα τα αρχικά τρεξίματα.
- Σε αυτό το σημείο, ο αριθμός των τρεξιμάτων έχει μειωθεί κατά M - 1.
- Εάν αυτός ο μειωμένος αριθμός διαδρομών εξακολουθεί να είναι μεγαλύτερος από ή ίσος με το M, γίνεται ένα άλλο πέρασμα, με τα τρεξίματα που δημιουργήθηκαν από το πρώτο πέρασμα ως είσοδο
  - Κάθε πέρασμα μειώνει τον αριθμό των διαδρομών με συντελεστή M - 1
- Τα περάσματα επαναλαμβάνονται όσες φορές απαιτείται, έως ότου ο αριθμός των διαδρομών είναι μικρότερος από M. ένα τελικό πέρασμα δημιουργεί τότε την ταξινομημένη έξοδο.

4) Map reduce on Very Large vs Very Small Datasets

Το Map Reduce ειναι για batch processing, αρα με μικρο dataset δεν αξιζει.

Θα υπάρξουν ακόμα πολλά overheads (κλήσεις συναρτήσεων) και δεν έχει νόημα και τελικά κάνει τον υπολογισμό αργό, γιατι ο,τι γλιτωνεις απο χρονο επεξεργασιας, το χανεις απο χρονο επικοινωνιας του cluster σε μικρα ντατασετ.

Επισης ισως οτι για το ιδιο chunk size μικρα datasets ισως δεν ειναι τοσο καλα μοιρασμένα στους worker nodes.

5) Lazy execution (τι είναι και πλεονεκτήματα-οφέλη)

με το lazy execution γράφεις σε υψηλότερο επίπεδο τις οδηγίες μετασχηματισμού, οι οποίες εκτελούνται on-demand (με το collect η κάποιο άλλο τρόπο)

Γλιτώνεις τον επεξεργαστικό φόρτο, ενώ παράλληλα μπορείς στο background να κανείς optimizations ώστε να βοηθήσεις τον planner στην επιλογή του optimal plan

Οι μετασχηματισμοί σε ένα RDD είναι lazy (ενδιάμεσες λειτουργίες), γιατί περιμένει να δώσεις και άλλους μετασχηματισμούς, για να δει αν μπορεί κάτι να το αποφύγει ή να το τρέξει παράλληλα. Η εκτέλεση γίνεται με την τερματική λειτουργία (action).

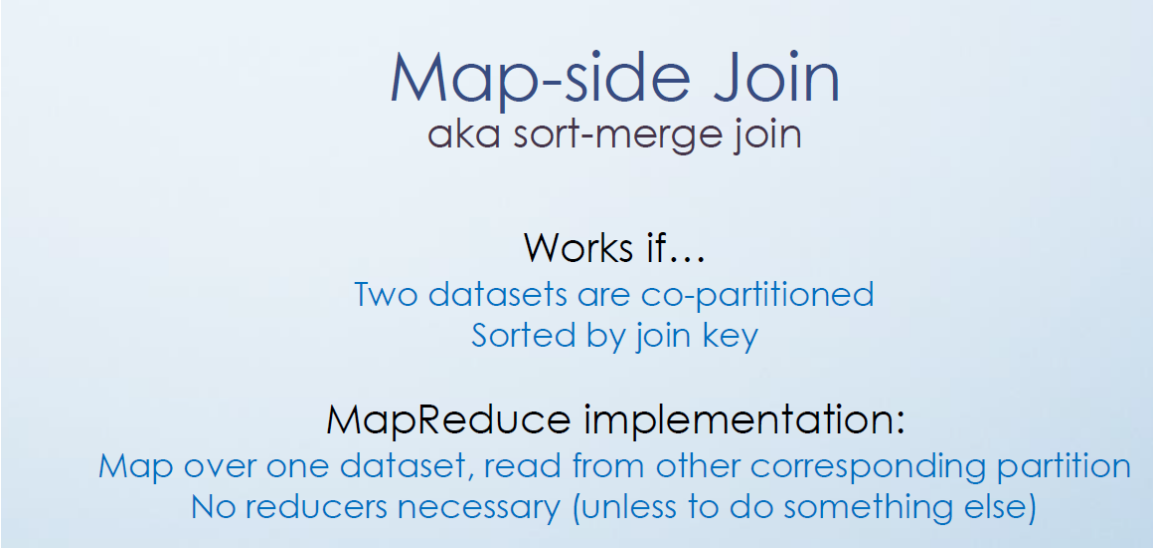
Lazy ➔ Δημιουργεί ευκαιρίες Βελτιστοποίησης

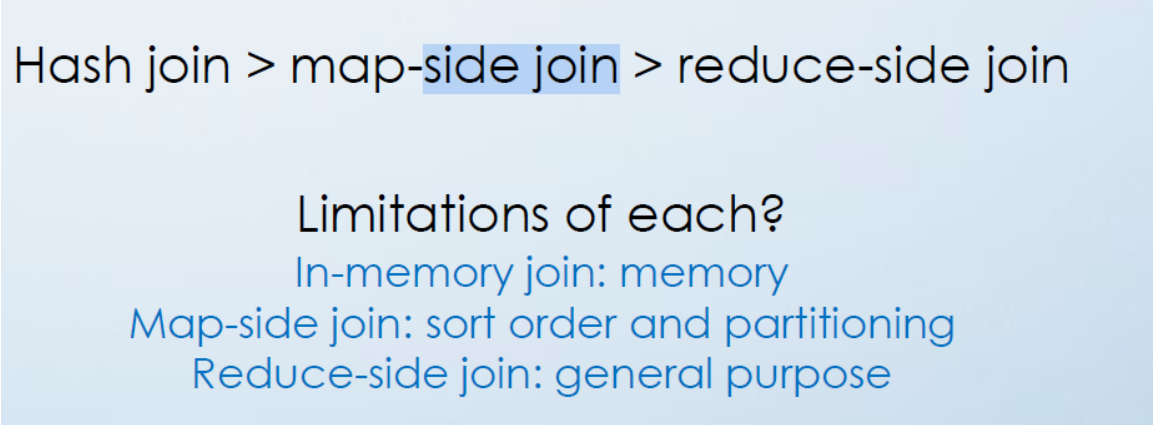
- **Βαριεστημένες λειτουργίες (laziness)**. Πολλές stream λειτουργίες όπως ***filtering***, ***mapping***, ή ***duplicate removal***, μπορούν να υλοποιηθούν lazily (βαριεστημένα), αναδεικνύοντας ευκαιρίες για βελτιστοποίηση. Οι λειτουργίες διαχωρίζονται σε **ενδιάμεσες** (stream--producing) λειτουργίες και **τερματικές** (value-- or side--effect--producing) λειτουργίες. ***Οι ενδιάμεσες λειτουργίες είναι πάντα lazy.***

6) Reduce Side Join (τι είναι, pros cons) σύγκριση με Map side Join

ReduceSide Join pros/cons: Αργο αλλά γενικου σκοπου (Οχι ταξινομημενα, Οχι Broadcast, Oxi cache)

Reduce side : > Map και στα δύο σύνολα δεδομένων > Εκπομπή πλειάδας ως τιμή με κλειδί συνένωσης ως ενδιάμεσο κλειδί > Το πλαίσιο εκτέλεσης συγκεντρώνει τις πλειάδες που μοιράζονται το ίδιο κλειδί > Εκτέλεση της συνένωσης στην πλευρά του reduce Παράλληλο sortmerge (ταξινόμηση-συγχώνευση) στις παραδοσιακές Βάσεις Δεδομένων (όχι ταξινομημένα, όχι broadcast, όχι cache)





MapSideJoin πλεονεκτηματα σε σχεση με ReduceSideJoin: Ταχυτητα, Δεν απαιτούνται reducers

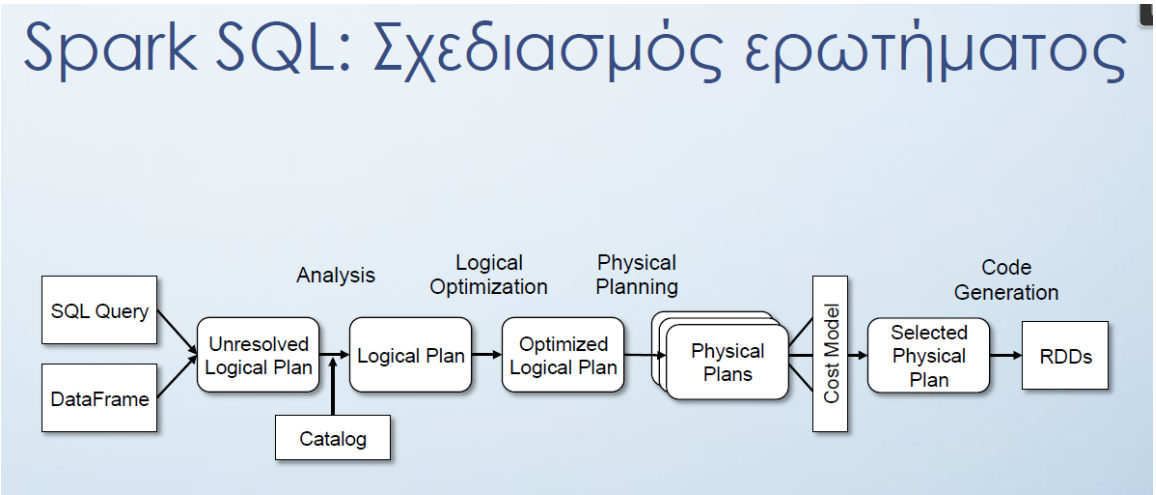
Το Map-Side είναι πιο γρήγορο αλλά προϋποθέτει ταξινομημένα datasets ως προς το κλειδί συνένωσης. * Το Reduce-Side είναι πιο αργό αλλά είναι γενικού σκοπού

7) Map Side Join δες Πανω

Λειτουργεί εάν δύο σύνολα δεδομένων είναι συν-διαμερισμένα και ταξινομημένα με το κλειδί συνένωσης. Map πάνω από το ένα σύνολο δεδομένων, ανάγνωση από άλλο αντίστοιχο διαμέρισμα Δεν απαιτούνται reducers (εκτός αν θέλουμε να κάνουμε κάτι άλλο)

8) SQL optimization ή query optimization

QueryOptimization (SQL): Query -> Parser -> Translator -> Σχεση -> Λογικο πλανο -> Optimize (Join, Select) -> Διαλέγει -> Φυσικο πλανο

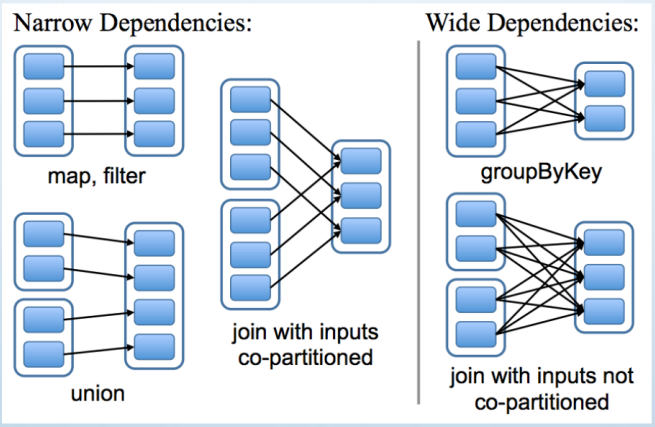


9) Narrow-Wide Dependencies

Narrow vs Wide dependencies: RDD -> εξαρταται μονο απο 1 parent RDD (Map) VS εξαρταται απο 1 ή περισσοτερα parent RDD (Reduce)

Σε Wide Dependencies γίνεται shuffle and sort

- Δυο ειδών αλληλεξαρτήσεις (dependencies)
  - wide (πλατιές) και narrow (στενές)
- Narrow dependencies
  - Το κάθε τμήμα του γονικού RDD χρησιμοποιείται από ***το πολύ ένα*** τμήμα του απόγονου RDD
- Wide dependencies
  - Το κάθε τμήμα του γονικού RDD χρησιμοποιείται από ***περισσότερα από ένα*** τμήματα του απόγονου RDD

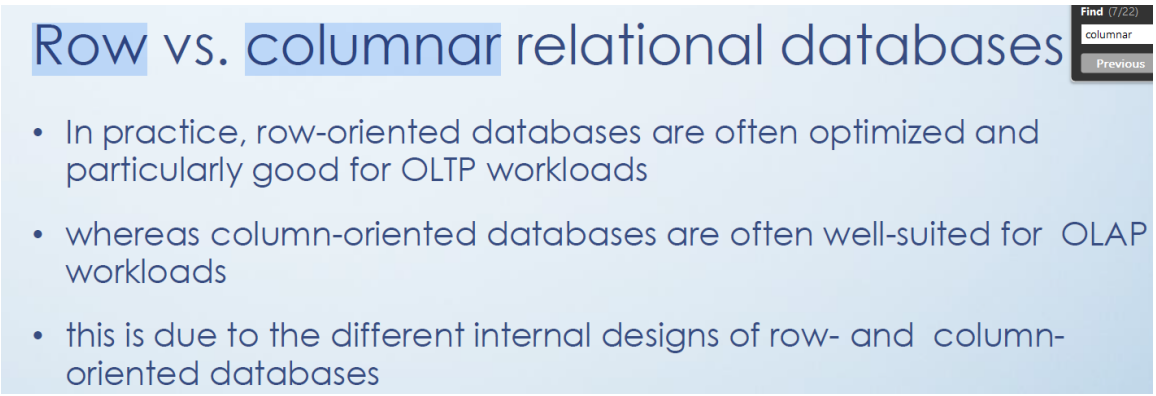


- 'Ενα map οδηγεί σε μια στενή εξάρτηση,
- 'Ενα join οδηγεί σε ευρείες αλληλεξαρτήσεις (εκτός αν οι γονείς είναι κατακερματισμένοι).
- **Wide:** θέλει shuffle και sort (δύσκολη υλοποίηση και αργό)
- **Narrow:** συμμετέχουν λίγοι workers (εύκολη υλοποίηση)

10) Columnal vs Row databases (query use cases)

Row vs. columnar relational databases

- All relational databases deal with tables, rows, and columns
- But there are sub-types:
  - row-oriented: they are internally organized around the handling of rows
  - columnar / column-oriented: these mainly work with columns
- Both types usually offer SQL interfaces and produce tables (with rows and columns) as their result sets
- Both types can generally solve the same queries
- Both types have specific use cases that they're good for (and use cases that they're not good for)



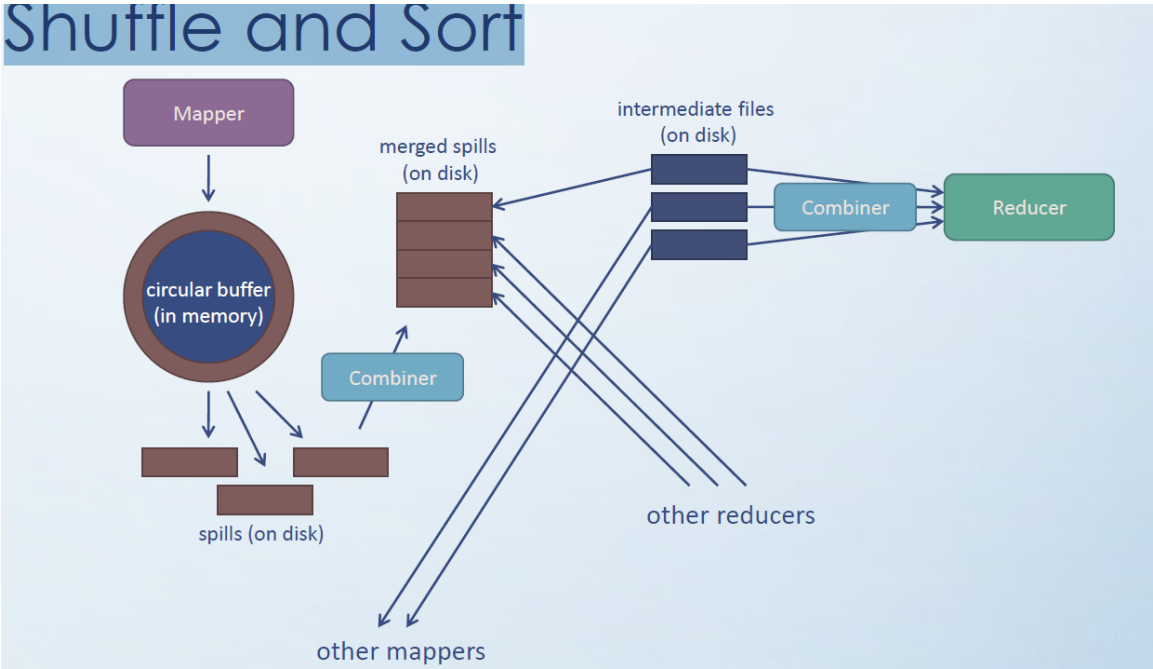
oltp einai grhgores allages
kai thes row
olap einai pio periploka aggregations
kai thes column

alla ta polu apla
diavazw grafw kai tetoia
einai grhgora me row

11) Shuffle & Sort in MapReduce

> **Shuffle**: Στέλνει όλα τα key-value records με το ίδιο key στον reducer που του υποδεικνύει ο partitioner,
> **Sort**: ταξινομεί όλα τα key-values σε έναν reducer βάσει του key.
> (Αν έχει οριστεί συνάρτηση combine τρέχει πριν το shuffle.)

Είναι ουσιαστικά ένας combiner , που μαζεύει όλα τα αποτελέσματα των partitions , και μετά τα σορταρει και τα μοιράζει στους reducers



Shuffle and Sort στο Hadoop

- Ίσως το ποιο περίπλοκο κομμάτι στο MapReduce
- Map μέρος
  - Οι εξόδοι του Map outputs διοχετεύονται στην μνήμη σε έναν κυκλικό απομονωτή buffer
  - Όταν ο απομονωτής γεμίσει, τα περιεχόμενα γίνονται spill (αδειάζουν) στον δίσκο
  - Τα Spills ενώνονται σε ένα μοναδικό, partitioned αρχείο (ταξινομημένο ανά κάθε partition): ο combiner τρέχει κατά την συνένωση
- Reduce μέρος
  - Καταρχάς τα map αποτελέσματα αντιγράφονται μέσω δικτύου στο μηχάνημα του reducer
  - Το "Sort" είναι ένα multi-pass merge στα δεδομένα του map (συμβαίνει στην μνήμη και στον δίσκο): ο combiner τρέχει κατά τα merges
  - Το τελικό merge πέρασμα πάει απευθείας στον reducer

12) Πώς γίνεται προσπέλαση σε εγγραφές σταθερού μήκους;

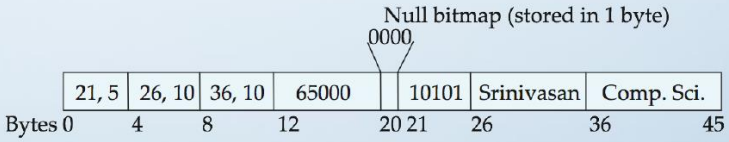
Η αποθηκευμένη εγγραφή i ξεκινά από το byte n * (i – 1), όπου n είναι το μέγεθος κάθε εγγραφής.

Εγγραφές μεταβλητού μεγέθους: Variable-Length Records

- Οι εγγραφές μεταβλητού μεγέθους προκύπτουν σε συστήματα βάσεων δεδομένων με διάφορους τρόπους:
  - Αποθήκευση πολλαπλών τύπων εγγραφών σε ένα αρχείο.
  - Τύποι εγγραφής που επιτρέπουν μεταβλητά μήκη για ένα ή περισσότερα πεδία όπως συμβολοσειρές (**varchar**)
  - Τύποι εγγραφών που επιτρέπουν επαναλαμβανόμενα πεδία (χρησιμοποιούνται σε ορισμένα παλαιότερα μοντέλα δεδομένων).
- Τρόποι αποθήκευσης εγγραφής μεταβλητού μήκους:
  - Οι εγγραφές σταθερού μήκους αποθηκεύονται με διαδοχική σειρά
  - Οι εγγραφές μεταβλητού μήκους αποθηκεύονται ως εξής:
    - Μεταδεδομένα για την τοποθεσία αποθήκευσης των πραγματικών δεδομένων εμφανίζονται στην αρχή της εγγραφής και αντιπροσωπεύονται από σταθερό μέγεθος (μετατόπιση, μήκος).
    - Τα πραγματικά δεδομένα αποθηκεύονται μετά από όλα τα πεδία σταθερού μήκους

Εγγραφές μεταβλητού μεγέθους: Variable-Length Records (συνέχεια)

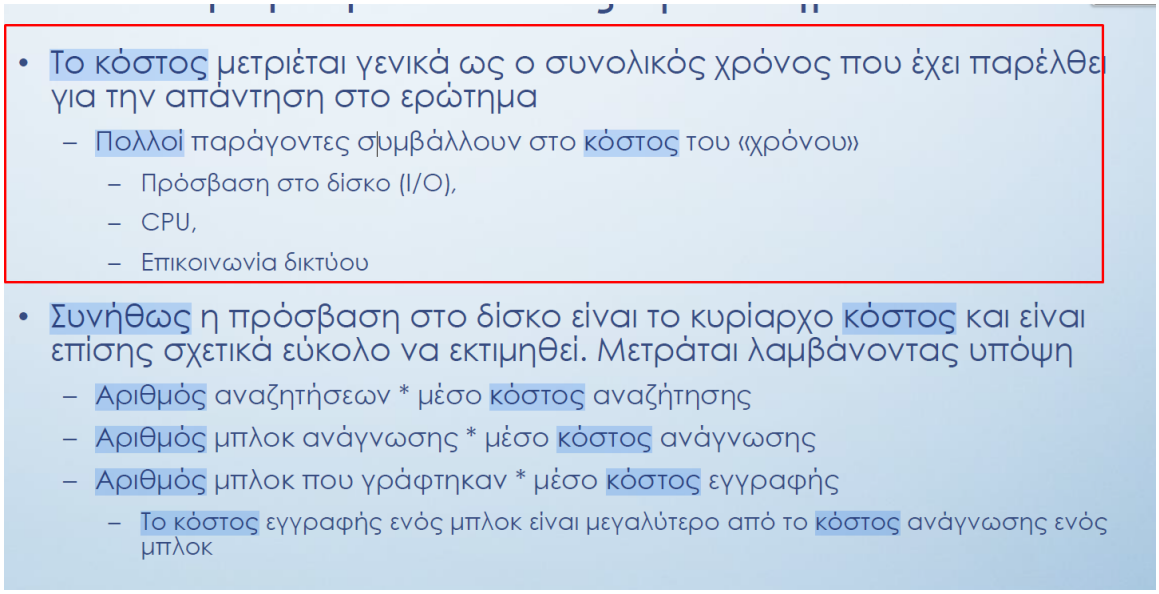
- Παράδειγμα εγγραφής εκπαιδευτή, το οποίο αποτελείται από 4 πεδία (ID, name, department, salary)
  - ID, name, department έχουν μεταβλητό μήκος
  - ο salary είναι σταθερού μήκους
- Διάταξη εγγραφής:



- Τα πρώτα 3 byte υποδεικνύουν πού αποθηκεύονται τα πρώτα 3 πεδία μεταβλητού μήκους και το μήκος κάθε πεδίου.
- Τα επόμενα σύνολα bytes χρησιμοποιούνται για την αποθήκευση του πεδίου σταθερού μεγέθους (μισθός).
- Τα null bitmap εξηγούνται στην επόμενη διαφάνεια

13) Πως γίνεται συνένωση αν οι πίνακες είναι ταξινομημένοι; - Δες το 3)

14) Τι συμβάλλει στο κόστος ενός sql ερωτήματος;



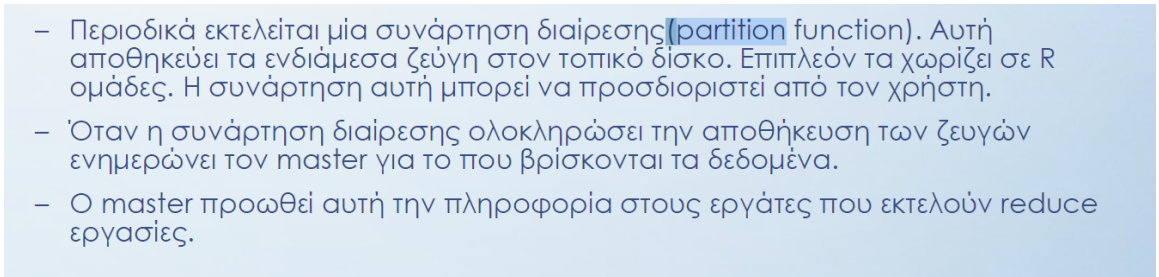
15) Map Reduce κοντά στα δεδομένα (data locality)

> Θέλουμε οι υπολογισμοί να γίνονται κοντά στα δεδομένα (ίδιο κόμβο π.χ.) ώστε να αποφεύγεται η μεταφορά μεγάλων δεδομένων εκεί που γίνεται ο υπολογισμός. Έτσι μικραίνει η συμφόρηση του δικτύου και αυξάνεται το throughput του συστήματος. Γίνονται spawn λιγότεροι executors.

προσπαθεις οι διεργασιες σου να βρισκονται physically κοντα στα δεδομενα που επεξεργαζονται

data locality = να στειλω την εργασια στα δεδομενα και όχι το αναποδο, μειωνοντας τη μεταφορα δεδομενων στο δικτυο, κανω δηλαδη την επεξεργασια πανω στα δεδομενα

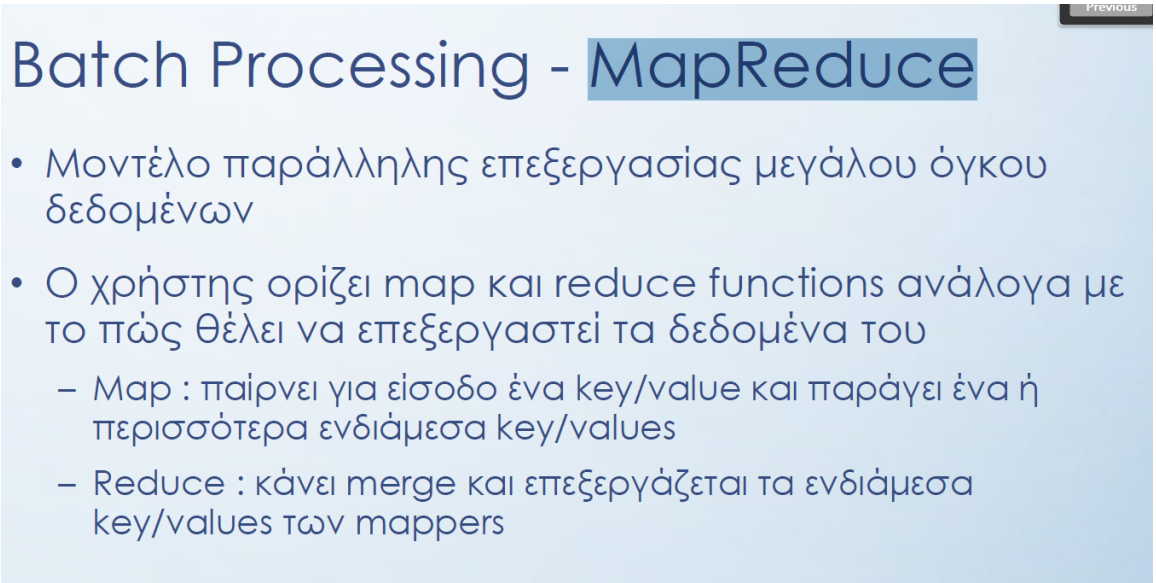
16) Τι είναι η partition function του MapReduce και ποια είναι η default στο Spark?

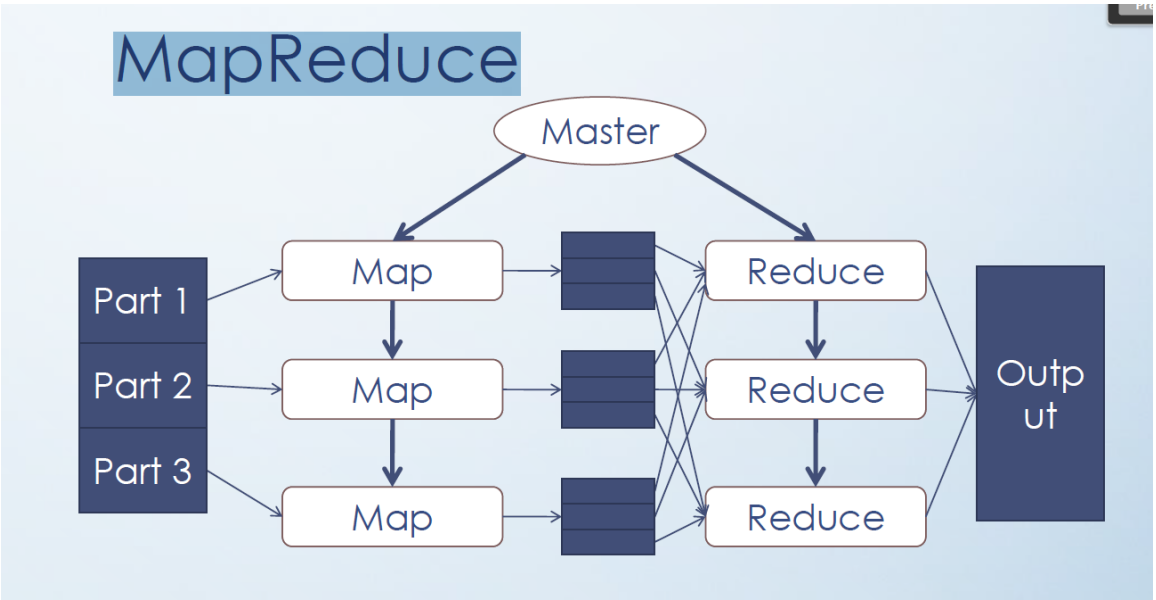


17) Τι συμβαίνει όταν αποτυγχάνει ένας Mapper, ενώ στέλνει τα δεδομένα σε έναν Reducer και κόβεται στη μέση η όλη διαδικασία αποστολής;

Επανεκτελειται αν δεν υπαρχουν αντιγραφα, αν υπαρχουν, speculative execution (οποιος προλαβει να τελειωσει πρωτος)

18) Πώς δουλεύει το MapReduce;





MapReduce

- Οι προγραμματιστές δηλώνουν δυο συναρτήσεις:
  map (k, v) → <k', v'>*
  reduce (k', v') → <k', v'>*
  - Όλες οι τιμές με το ίδιο κλειδί συνενώνονται μαζί
- Το πλαίσιο εκτέλεσης αναλαμβάνει όλα τα άλλα…
- Όχι ακριβώς,…συνήθως, οι προγραμματιστές επίσης δηλώνουν:
  partition (k', number of partitions) → partition for k'
  - Συνήθως ένα απλό hash του κλειδιού, πχ hash(k') mod n
  - Σπάει το εύρος τιμών των κλειδιών για παράλληλες εκτελέσεις reduce
  combine (k', v') → <k', v'>*
  - Μινι-reducers που τρέχουν στην μνήμη τοπικά μετά την φάση του Map
  - Χρησιμοποιείται σαν βελτιστοποίηση για να μειώσουν την κίνηση στο δίκτυο

MapReduce

- Το πρόβλημα "σπάει" σε 2 φάσεις, την Map και την Reduce
- **Map**: Μη αλληλο-επικαλυπτόμενα κομμάτια από δεδομένα εισόδου (εγγραφές <key,value>) ανατίθενται σε διαφορετικές διεργασίες (mappers) οι οποίες βγάζουν ένα σετ από ενδιάμεσα <key,value> αποτελέσματα
- **Reduce**: Τα δεδομένα της Map φάσης τροφοδοτούνται σε ένα συνήθως μικρότερο αριθμό διεργασιών (reducers) οι οποίες "συνοψίζουν" τα αποτελέσματα εισόδου σε μικρότερο αριθμό <key,value> εγγραφών

19) HDFS γιατί έχει υψηλή διαθεσιμότητα;

Κρατάει τα blocks της μνήμης σε παραπάνω από ένα κόμβο, κρατάει αντίγραφα κάθε αρχείου σε deafault=3 nodes

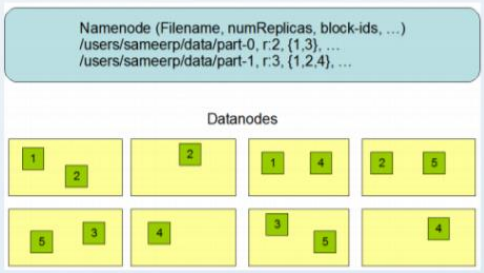
- Γρήγορη ανάρρωση από σφάλματα

- Αντιγραφή

της κατάστασης του Master

20) Πώς εξασφαλίζεται το fault tolerance se HDFS?

NameNode αντίγραφα Blocks



- **Η στρατηγική που ακολουθείται**
  - Ένα αντίγραφο στον τοπικό κόμβο.
  - Δεύτερο αντίγραφο στο ίδιο rack
  - Τρίτο αντίγραφο σε απομακρυσμένο rack
  - Επιπλέον αντίγραφα τοποθετούνται σε τυχαίους κόμβους
- **Οι Clients διαβάζουν από τον πλησιέστερο αντίγραφο**

Ανοχή στα σφάλματα

- Ο master επικοινωνεί με τους εργάτες περιοδικά. Εάν κάποιος δεν ανταποκριθεί για ένα χρονικό διάστημα τότε αναθέτει την εργασία του σε κάποιον άλλο.
- Ο master επιπλέον περιοδικά διατηρεί αντίγραφα ελέγχου των δομών του. Σε περίπτωση βλάβης ένας νέος μπορεί να ξεκινήσει άμεσα.
- Τα ενδιάμεσα αποτελέσματα που παράγονται από τις map και reduce εργασίες διατηρούνται σε προσωρινά αρχεία σε τοπικά συστήματα αρχείων έως όλη η είσοδος να έχει επεξεργαστεί. Στη συνέχεια ενημερώνεται ο master και η πληροφορία γίνεται διαθέσιμη σε όλους.

21) Ποιοί παράγοντες μας νοιάζουν στην βελτιστοποίηση;

- Βελτιστοποίηση
  - Εύρεση ενός **πλάνου υπολογισμού**
  - Αναζήτηση βέλτιστου πλάνου (μεταξύ πολλών πιθανών)

- **Βελτιστοποίηση ερωτήματος**: Ανάμεσα σε όλα τα ισοδύναμα πλάνα υπολογισμού επέλεξε αυτό με το χαμηλότερο κόστος.
  - Το κόστος υπολογίζεται χρησιμοποιώντας στατιστικές πληροφορίες από τον κατάλογο βάσεων δεδομένων
    - Για παράδειγμα, αριθμός πλειάδων σε κάθε σχέση, μέγεθος πλειάδων κ.λπ.

- Βήματα στη **Βελτιστοποίηση Ερωτήσεων που βασίζεται στο κόστος***
  1. Δημιουργία λογικά ισοδύναμων δέντρων ερωτήσεων με την χρήση **κανόνων μετασχηματισμού**
  2. Μετέτρεψε το ισοδύναμο δέντρο σε σχέδιο εκτέλεσης με την επιλογή αλγορίθμου εκτέλεσης
  3. Επέλεξε το φτηνότερο σχέδιο εκτέλεσης με βάση το **εκτιμώμενο κόστος**
- Εκτίμηση του κόστους του σχεδίου με χρήση:
  - Στατιστικής πληροφορίας για τις σχέσεις.
    - Αριθμός πλειάδων, αριθμός μοναδικών τιμών για ένα πεδίο, κλπ.
  - Εκτίμηση στατιστικών για ενδιάμεσα αποτελέσματα
    - Για τον υπολογισμό κόστους περίπλοκων εκφράσεων
  - Τύπος υπολογισμού των αλγορίθμων βασισμένος σε στατιστικά

22) Πώς γίνεται distributed join με ένα μικρό και ένα μεγάλο dataset? Δες Hash και Broadcast 2) ???Ή ΟΧΙ? BroadcastHashJoin (== hash Join) εφόσον το μικρό χωράει στην μνήμη.

23) Διαφορές GFS – HDFS

HDFS is open source GFS

GFS and **HDFS are similar in many aspects** and are used for storing large amounts of data sets. HDFS being a module of an open source project (Hadoop) it is vastly applicable (Yahoo!, Facebook, IBM etc use HDFS) as compared to the proprietary GFS. MapReduce provides distributed, scalable and data- intensive computing. 9 Σεπ 2014

Hadoop Distributed File System HDFS	Google File System GFS
Cross Platform	Linux
Developed in Java environment	Developed in c,c++ environment
At first its developed by Yahoo and now its an open source Framework	Its developed by Google
It has Name node and Data Node	It has Master-node and Chunk server
128 MB will be the default block size	64 MB will be the default block size
Name node receive heartbeat from Data node	Master node receive heartbeat from Chunk server
Commodities hardware were used	Commodities hardware werused
WORM – Write Once and Read Many times	Multiple writer , multiple reader model
Deleted files are renamed into particular folder and then it will removed via garbage	Deleted files are not reclaimed immediately and are renamed in hidden name space and it will deleted after three days if it's not in use
No Network stack issue	Network stack Issue
Journal ,editlog	Oprational log
only append is possible	random file write possible

24) Τι είναι οι Combiners?

Βρίσκονται μετά το map (πριν την partition) και λειτουργούν σαν ένα mini-reduce τοπικά στη μνήμη, σαν βελτιστοποίηση για να μειώσουν την κίνηση στο δίκτυο. Ουσιαστικά, συνοψίζουν τα outputs με ίδιο key.

25) Map vs Reduce side δες 6)

Ποιο είναι πιο γρήγορο map side reduce side : Map αλλα με προυποθεσεις (Sort)

26) Spark Hash vs Sort shuffle

Το Hash Shuffle δημιουργεί 1 αρχείο για κάθε reducer, ενώ με το sort shuffle εξάγεται ένα μόνο αρχείο που έχει δεδομένα ταξινομημένα ανά "reducer" id και είναι ευρετηριασμένο.
Για μικρό αριθμό reducers το hash shuffle είναι πιο γρήγορο, γι'αυτό χρησιμοποιείται ως κύριο το sort shuffle και σαν εφεδρικό το hash shuffle όταν ο αριθμός reducers είναι μικρότερος από ένα κατώφλι (default 200).
Αν στο sort shuffle δεν έχω αρκετή μνήμη να χωρέσω όλο το map output τότε spill δεδομένα στο δίσκο.