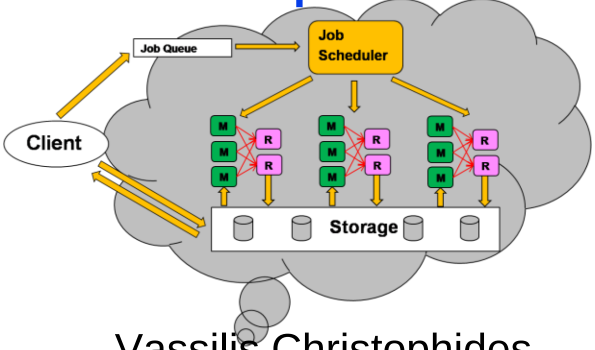
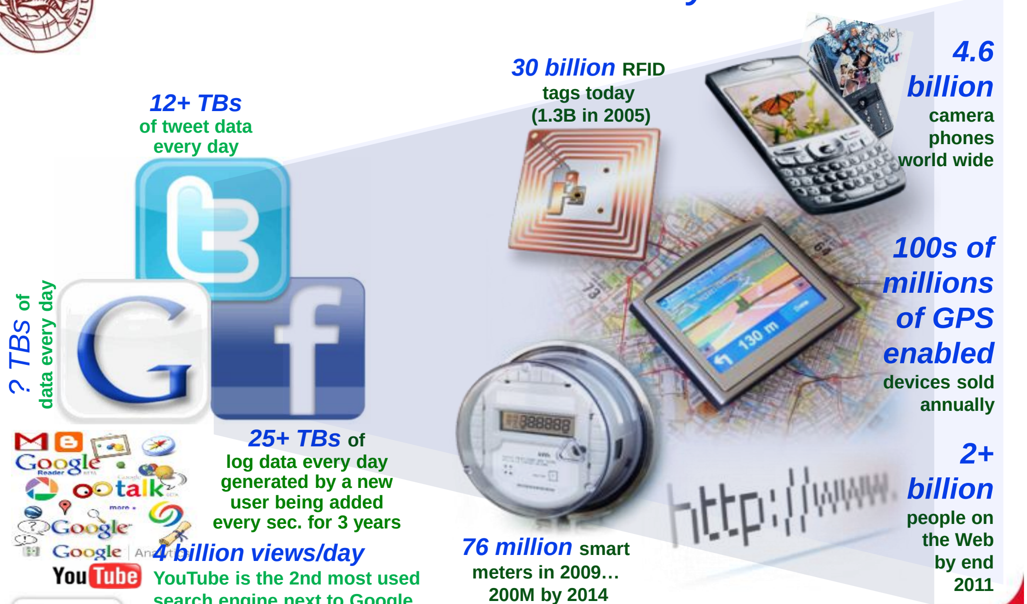
Relational Data Processing on MapReduce



Vassilis Christophides
christop@csd.uoc.gr
http://www.csd.uoc.gr/~hy562
University of Crete, Fall 2019

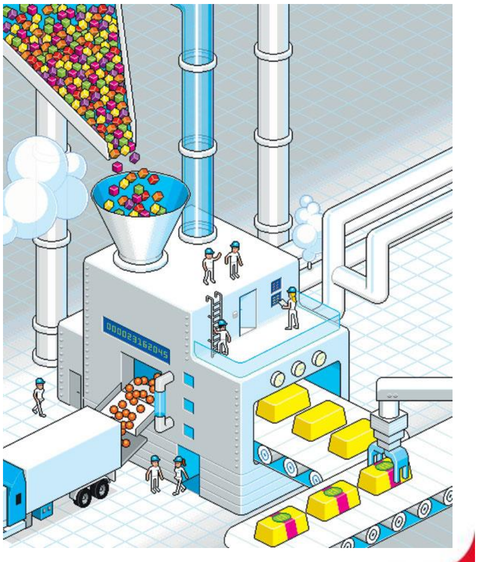
Peta-scale Data Analysis



Big Data Analysis

Fall 2019

- A lot of these datasets have some structure
  - Query logs
  - Point-of-sale records
  - User data (e.g., demographics)
  - …
- How do we perform data analysis at scale?
  - Relational databases and SQL
  - MapReduce (Hadoop)



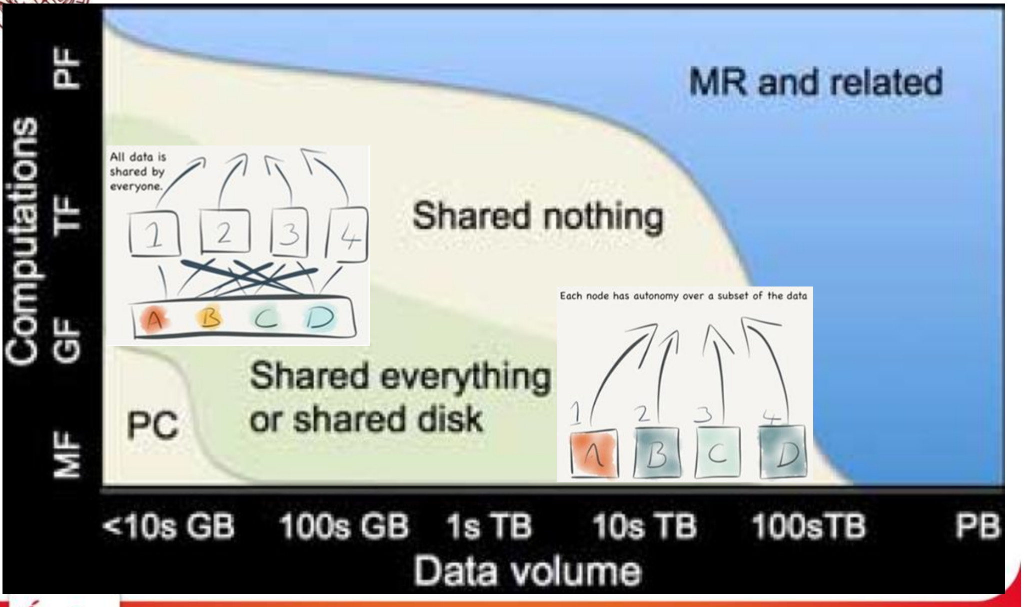
Relational Databases vs. MapReduce

Fall 2019

- Relational databases:
  - Multipurpose: analysis and transactions; batch and interactive
  - Data integrity via ACID transactions
  - Lots of tools in software ecosystem (for ingesting, reporting, etc.)
  - Supports SQL (and SQL integration, e.g., JDBC)
  - Automatic SQL query optimization
- MapReduce (Hadoop):
  - Designed for large clusters, fault tolerant
  - Data is accessed in “native format”
  - Supports many query languages
  - Programmers retain control over performance

Fall 2019

Parallel Computation & Data Size Matters!

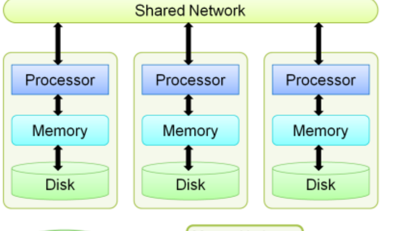


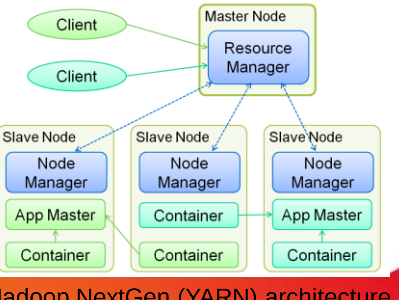
Fall 2019

Parallel Relational Databases vs. MapReduce

- Parallel relational databases
  - Schema on “write”
  - Failures are relatively infrequent
  - “Possessive” of data
  - Mostly proprietary
- MapReduce
  - Schema on “read”
  - Failures are relatively common
  - In situ data processing
  - Open source

Shared-nothing architecture for parallel processing





Hadoop NextGen (YARN) architecture

Fall 2019

MapReduce: A Major Step Backwards?

- MapReduce is a step backward in database access
  - ◆Separation of the schema from the application is good
    - Sharing across multiple MR programs is difficult
  - ◆Declarative access languages are good
    - Does not requires highly-skilled programmers
- MapReduce is poor implementation
  - ◆Brute force and only brute force
    - no indexes: Wasteful access to unnecessary data
  - ◆Don’t need 1000 nodes to process petabytes
    - Parallel DBs do it in fewer than 100 nodes
- MapReduce is missing features
  - ◆Bulk loader, indexing, updates, transactions...
  - ◆No support for JOINs:
    - Requires multiple MR phases for the analysis

Agrawal et al., VLDB 2010 Tutorial

Fall 2019

Map Reduce vs Parallel DBMS

	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	✓

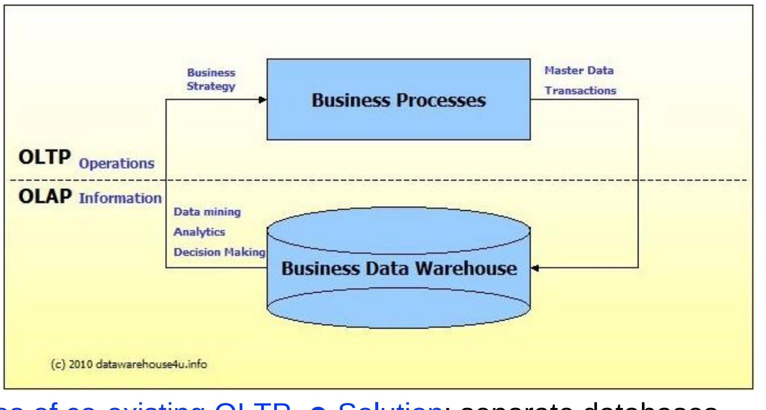
[Pavlo et al., SIGMOD 2009, Stonebraker et al., CACM 2010, …]

Database Workloads

- OLTP (online transaction processing)
  - Typical applications: e-commerce, banking, airline reservations
  - User facing: *real-time*, *low latency*, *highly-concurrent*
  - Tasks: relatively small set of "standard" *transactional queries*
  - Data access pattern: *random reads*, *updates*, *writes* (involving relatively small amounts of data)

- OLAP (online analytical processing)
  - Typical applications: business intelligence, data mining
  - Back-end processing: *batch workloads*, *less concurrency*
  - Tasks: complex *analytical queries*, often ad hoc
  - Data access pattern: *table scans*, large amounts of data involved per query

One Database or Two?



- Downsides of co-existing OLTP and OLAP workloads
  - *Poor memory management*
  - *Conflicting data access patterns*
  - *Variable latency*
- Solution: separate databases
  - *User-facing OLTP database* for high-volume transactions
  - *Data warehouse for OLAP workloads*
  - How do we *connect the two*?

OLTP/OLAP Integration

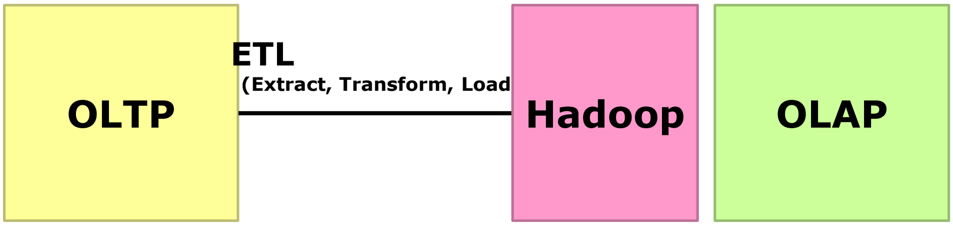
- OLTP database for user-facing transactions
  - Retain records of all activity
  - Periodic ETL (e.g., nightly)
- Extract-Transform-Load (ETL)
  - Extract records from source
  - Transform: clean data, check integrity, aggregate, etc.
  - Load into OLAP database
- OLAP database for data warehousing
  - Business intelligence: reporting, ad hoc queries, data mining, etc.
  - Feedback to improve OLTP services

OLTP/OLAP Architecture: Hadoop?

What about here?

Hadoop here?

OLTP/OLAP/Hadoop Architecture

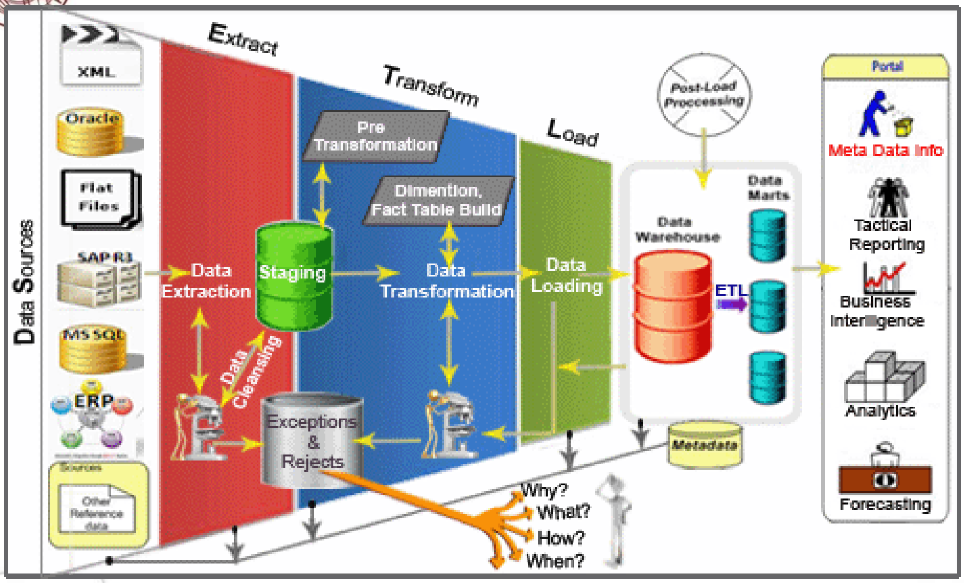


- Why does this make sense?

ETL Bottleneck

- Reporting is often a nightly task:
  - ETL is often slow (see next picture)!
    - What happens if processing 24 h of data takes longer than 24 h?
- Often, with noisy datasets, ETL is the analysis!
  - ETL necessarily involves brute force data scans: L, then E and T?
- Hadoop is perfect:
  - Most likely, you already have some data warehousing solution
  - *Ingest is limited by speed of HDFS*
  - *Scales out* with more nodes
  - *Massively parallel* and much cheaper than parallel databases
  - Ability to use *any processing tool*
  - ETL is a *batch process* anyway!

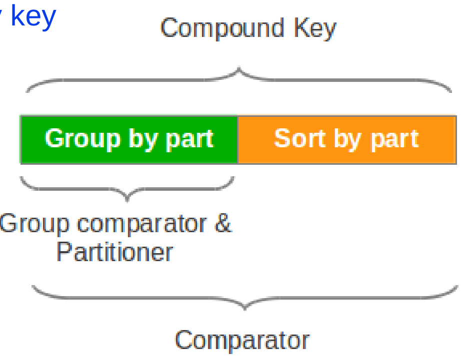
A Closer Look at ETL



MapReduce Algorithms for Processing Relational Data

Secondary Sorting

- MapReduce sorts input to reducers by key
  - Values are arbitrarily ordered
- What if want to sort value also?
  - E.g., k → `(v1, R), (v3, R), (v4, R), (v8, R)…`
- Solution 1:
  - Buffer values in memory, then sort
  - Why is this a bad idea?
- Solution 2:
  - "Value-to-key conversion": extends the key with part of the value
  - Let execution framework do the sorting
  - Preserve state across multiple key-value pairs to handle processing
  - Anything else we need to do?



Value-to-Key Conversion

Before

`k → (v1, R), (v4, R), (v8, R), (v3, R)…`

Values arrive in arbitrary order…

After

```
(k, v1) → (v1, R)
(k, v3) → (v3, R)
(k, v4) → (v4, R)
(k, v8) → (v8, R)
     …
```

Values arrive in sorted order…
Process by preserving state across multiple keys!

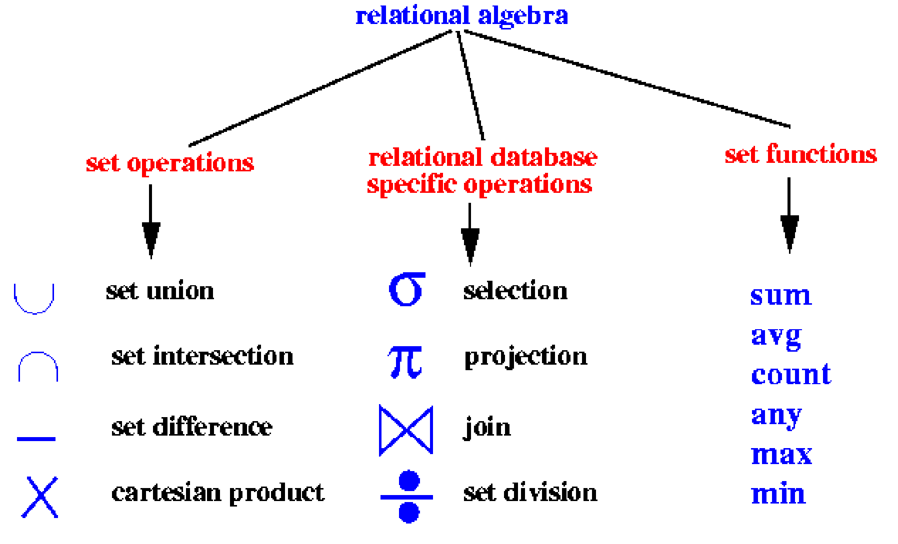
- Default `comparator`, group `comparator`, and `Partitioner` has to be tuned to use the appropriate part of the key

Working Scenario

- Two tables:
  - User demographics (gender, age, income, etc.)
  - User page visits (URL, time spent, etc.)

- Analyses we might want to perform:
  - Statistics on demographic characteristics
  - Statistics on page visits
  - Statistics on page visits by URL
  - Statistics on page visits by demographic characteristic
  - …

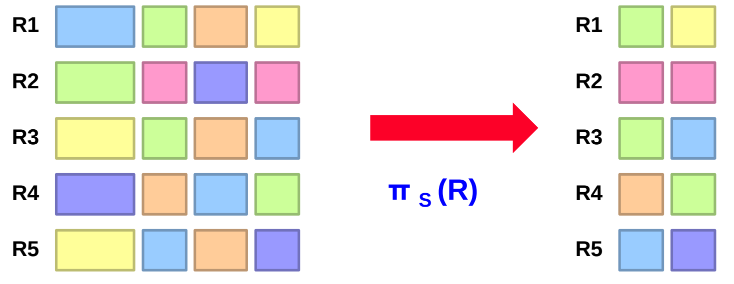
Relational Algebra



www.mathcs.emory.edu/~cheung/Courses/377/Syllabus/4-RelAlg/intro.html

Projection

Fall 2019



22

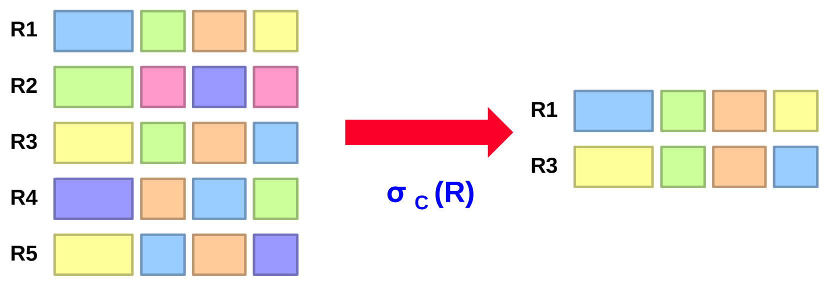
Projection in MapReduce

Fall 2019

- Easy!
  - Map over tuples, emit new tuples with the projected attributes
    - For each tuple t in R, construct a tuple t' by eliminating those components whose attributes are not in S, emit a key/value pair (t', t')
  - No reducers (reducers are the *identity* function), unless for regrouping or resorting tuples
    - the Reduce operation performs duplicate elimination
  - Alternatively: perform in reducer, after some other processing
- Basically limited by HDFS streaming speeds
  - Speed of *encoding/decoding* tuples becomes important
  - Relational databases take advantage of *compression*
  - Semi-structured data? No problem!

23

Selection



Selection in MapReduce

- Easy!
  - Map over tuples, emit only tuples that meet selection criteria
    - For each tuple t in R, check if t satisfies C and If so, emit a key/value pair (t, t)
  - No reducers (reducers are the *identity* function), unless for regrouping or resorting tuples
  - Alternatively: perform in reducer, after some other processing

- Basically limited by HDFS streaming speeds:
  - Speed of *encoding/decoding tuples* becomes important
  - Relational databases take advantage of *compression*
  - Semistructured data? No problem!

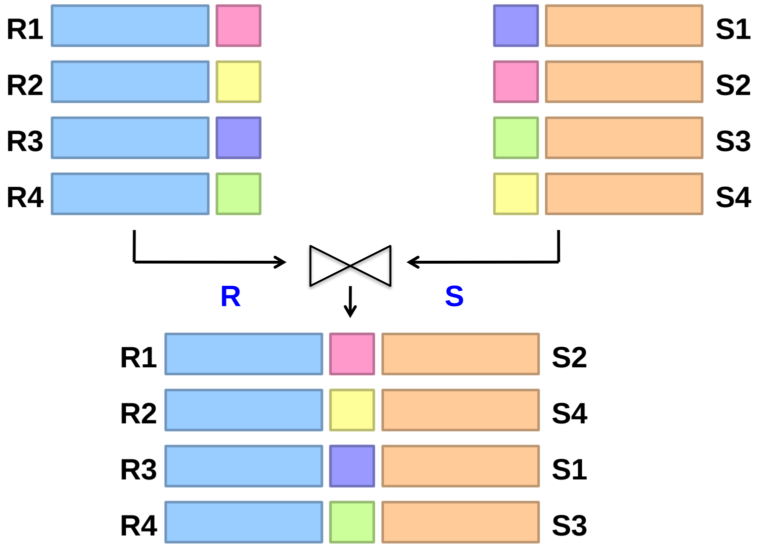
Set Operations in Map Reduce

- R(X,Y) ∪ S(Y,Z)
  - Map: for each tuple t either in R or in S, emit (t,t)
  - Reduce: either receive (t,[t,t]) or (t,[t])
    - Always emit (t,t)
    - We perform duplicate elimination
- R(X,Y) ∩ S(Y,Z)
  - Map: for each tuple t either in R or in S, emit (t,t)
  - Reduce: either receive (t,[t,t]) or (t,[t])
    - Emit (t,t) in the former case and nothing (t, NULL) in the latter
- R(X,Y) - S(Y,Z)
  - Map: for each tuple t either in R or in S, emit (t, R or S)
  - Reduce: receive (t,[R]) or (t,[S]) or (t,[R,S])
    - Emit (t,t) only when received (t,[R]) otherwise nothing (t, NULL)

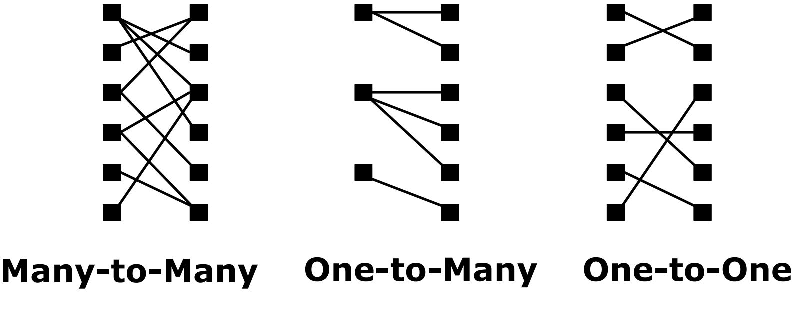
Group by… Aggregation

- Example: What is the average time spent per URL?
- In SQL:
  - `SELECT url, AVG(time) FROM visits GROUP BY url`
- In MapReduce: Let R(A, B, C) be a relation to which we apply $\gamma_{A,\theta(B)}(R)$
  - The map operation prepares the grouping (e.g., emit time, keyed by url)
  - The grouping is done by the framework
  - The reducer computes the aggregation (e.g. average)
  - Eventually, optimize with combiners
  - Simplifying assumptions: *one grouping attribute* and *one aggregation* function

Relational Joins



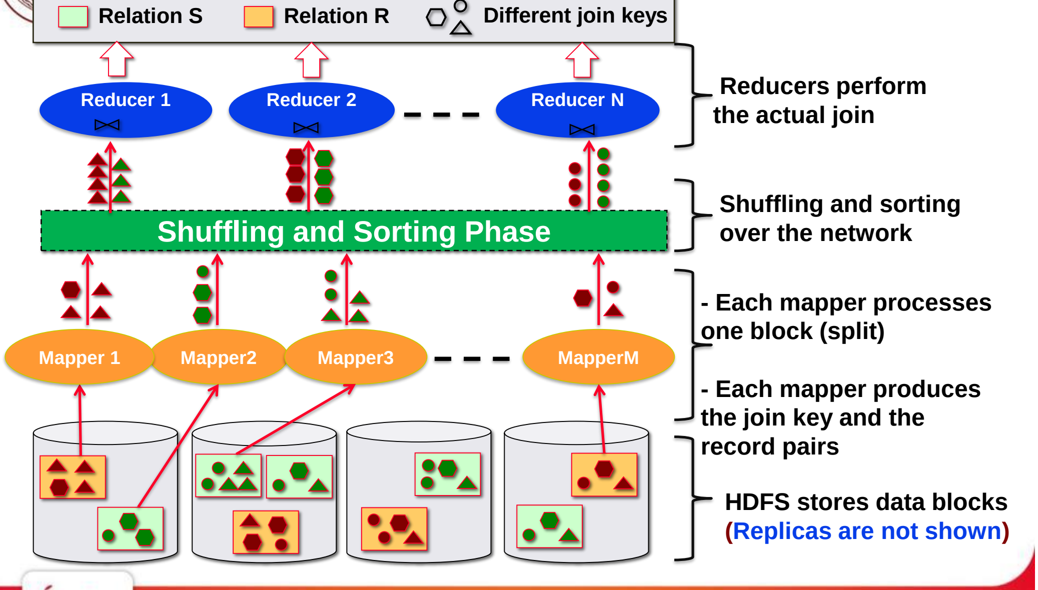
Types of Relationships



Join Algorithms in MapReduce

- "Join" usually just means equi-join, but we also want to support *other* join predicates
- Hadoop has some built-in join support, but our goal is to understand important algorithm design principles
- Algorithms
  - Reduce-side join
  - Map-side join
  - In-memory join
    - Striped variant
    - Memcached variant

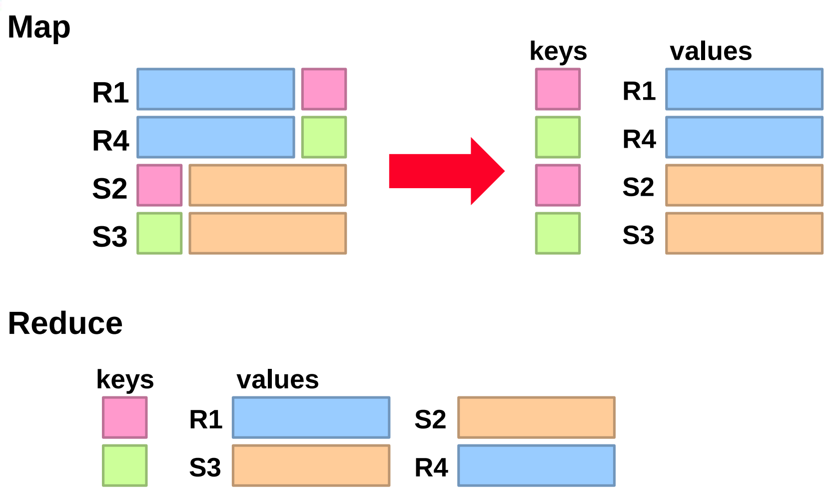
Re-Partition Join



Reduce-side Join

- Basic idea: group by join key
  - Execution framework brings together tuples sharing the same key
  - Similar to a “sort-merge join” in the database terminology
- A map function
  - Receives a record in R and S
  - Emits its join attribute value as a key and the record as a value
- A reduce function
  - Receives each join attribute value with its records from R and S
  - Perform actual join between the records in R and S
- Two variants
  - 1-to-1 joins
  - 1-to-many and many-to-many joins

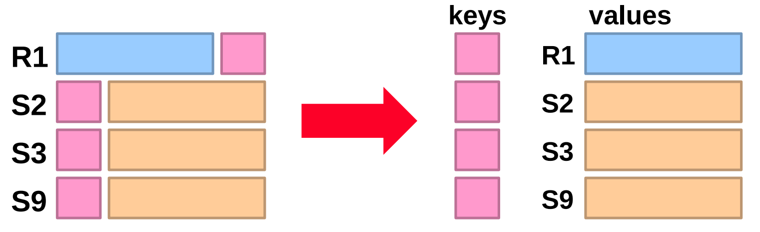
Reduce-side Join: 1-to-1



Note: no guarantee if R is going to come first or S!

Reduce-side Join: 1-to-Many

Map

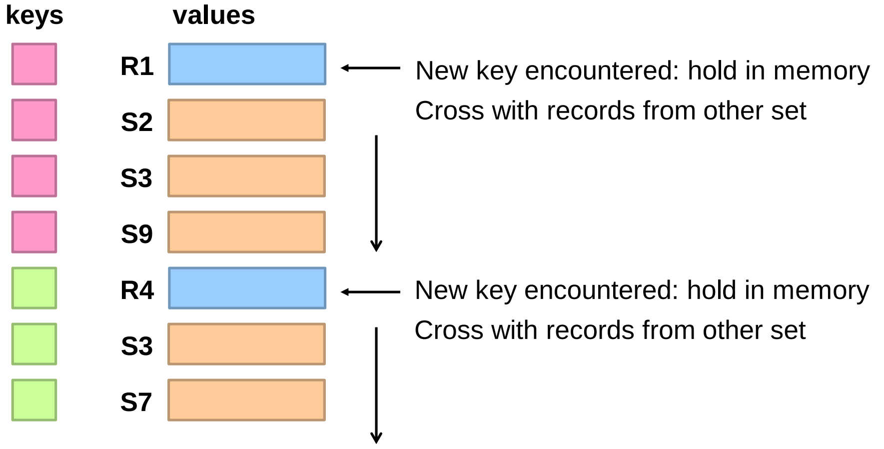


Reduce

- What's the problem?
  - R is the one side, S is the many

Reduce-side Join: Value-to-Key Conversion

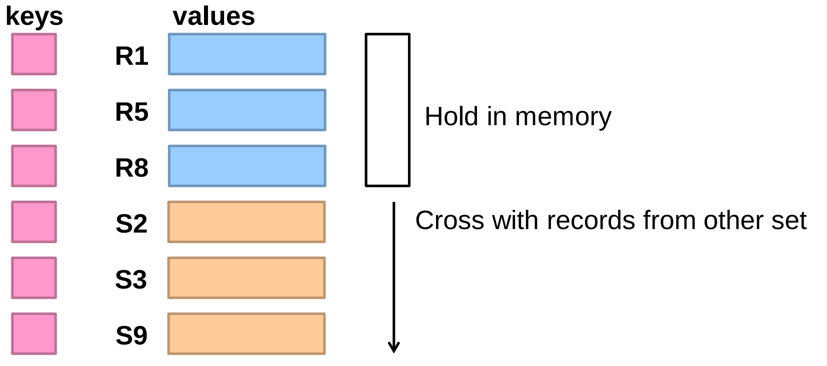
In reducer… Buffer all values in memory, pick out the tuple from R, and then cross it with every tuple from S to perform the join



www.inkling.com/read/hadoop-definitive-guide-tom-white-3rd/chapter-8/example-8-9

Reduce-side Join: Many-to-Many

In reducer…

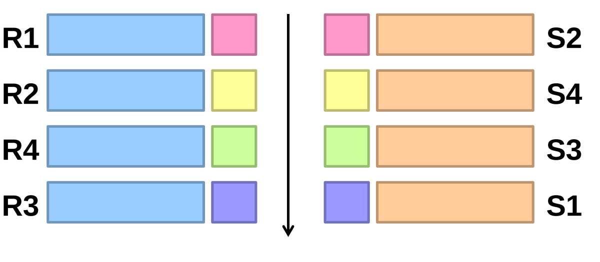


- What's the problem?
  - R is the smaller dataset

Map-side Join: Basic Idea

- What are the limitations of reduce-side joins?
  - Both relations are transferred over the network

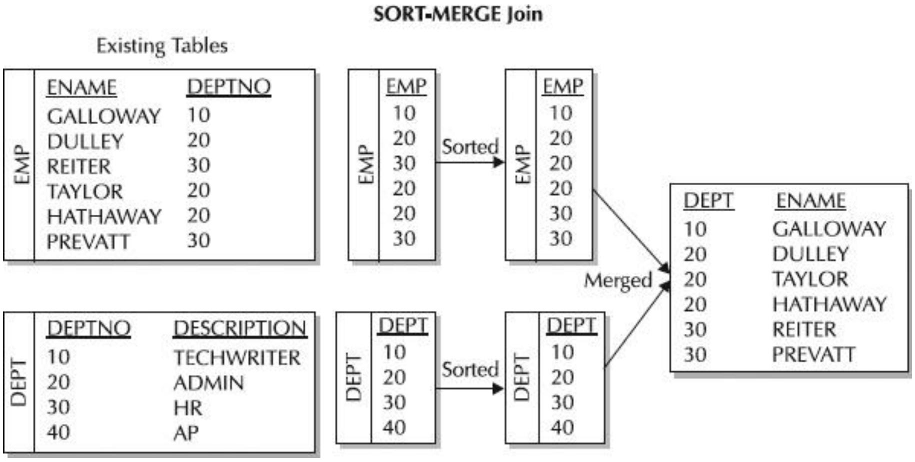
- Assume two datasets are sorted by the join key:



A sequential scan through both relations to join:
called a "sort-merge join" in database terminology

Map-side Join: Parallel Scans

- If datasets are sorted by join key, join can be accomplished by a scan over both relations



https://logicalread.com/oracle-11g-sort-merge-joins-mc02/

Map-side Join: Parallel Scans

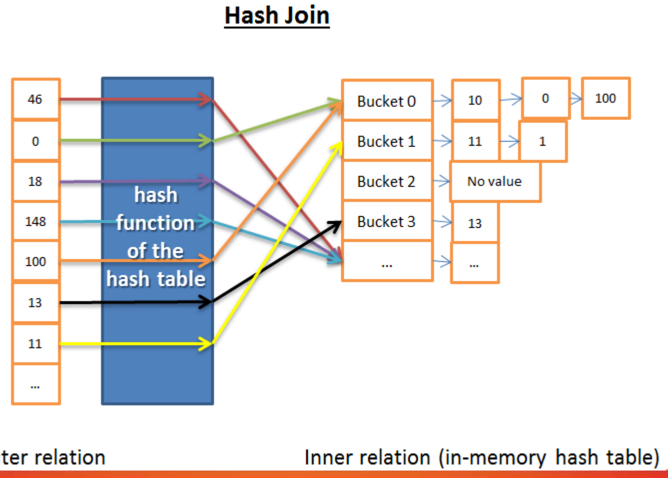
- How can we accomplish this in parallel?
  - Partition and sort both relations in the same manner

- In MapReduce:
  - Map over one relation, read from other corresponding partition
  - No reducers necessary (unless to repartition or resort)

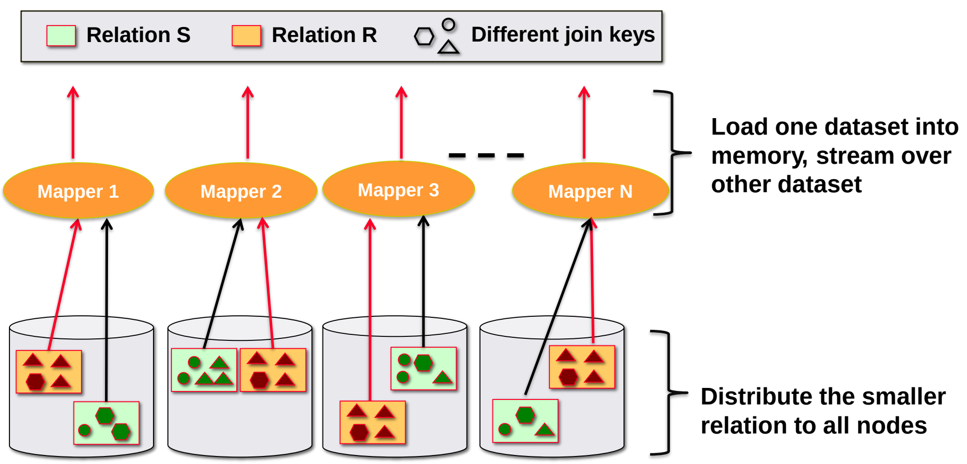
- Consistently partitioned relations: realistic to expect?
  - Depends on the workflow
  - For ad hoc data analysis, reduce-side are more general, although less efficient

In-Memory Join: Variants

- Basic idea: load one dataset into memory, stream over other dataset
  - Works if R << S and R fits into memory
  - Called a "hash join" in database terminology



Broadcast/Replication Join



In-Memory Join: Variants

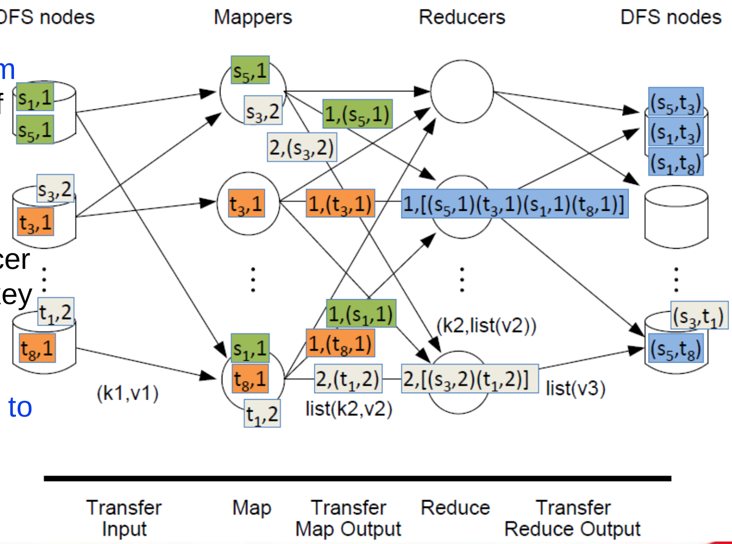
- MapReduce implementation
  - Distribute R to all nodes
  - Map over S, each mapper loads R in memory, hashed by join key
  - For every tuple in S, look up join key in R
  - No reducers, unless for regrouping or resorting tuples
- Downside: need to copy R to all mappers
  - Not so bad, since R is small

Which Join to Use?

- In-memory join > map-side join > reduce-side join
  - Why?
- Limitations of each?
  - In-memory join: memory
  - Map-side join: sort order and partitioning
  - Reduce-side join: general purpose algorithm but sensible to data skewness?
- What about non-equi joins?
  - Inequality (S.A<R.A): map just forwards R-tuples, but replicates S-tuples for all larger R.A values as keys

Problems With Standard Repartition Equi-Joins

- Degree of parallelism limited by number of distinct join values
- Data skew
  - ◆ If one join value dominates, reducer processing that key will become bottleneck
- Does not generalize to other joins



Standard Repartition Equi-Join Algorithm

- Consider only the pairs with the same join attribute values

		s_1	s_2	s_3	s_4	s_5	s_6
		a_1	a_1	a_2	a_2	a_3	a_4
r_1	a_1	O	O				
r_2	a_1	O	O				
r_3	a_2			O	O		
r_4	a_3					O	

Naïve join algorithm

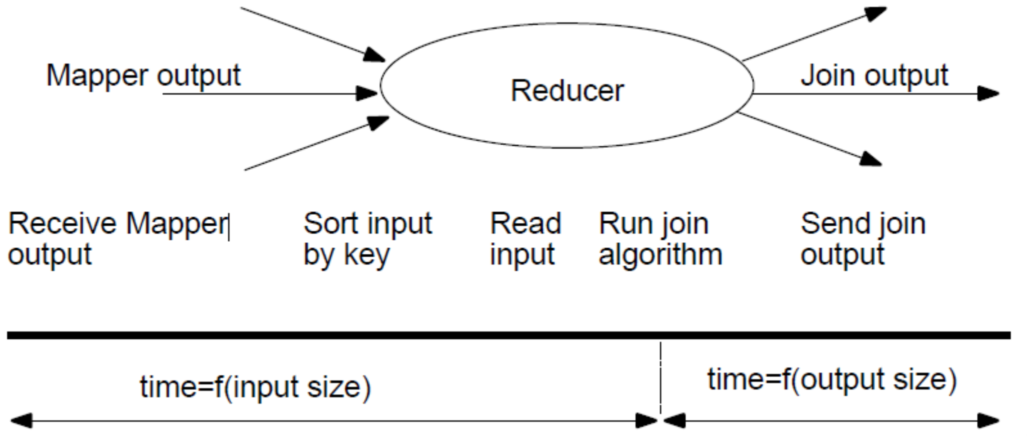
enumerated pairs

Standard repartition join algorithm

		s_1	s_2	s_3	s_4	s_5	s_6
		a_1	a_1	a_2	a_2	a_3	a_4
r_1	a_1	O	O				
r_2	a_1	O	O				
r_3	a_2			O	O		
r_4	a_3					O	

Reducer-Centric Cost Model

- Difference between join implementations starts with Map output

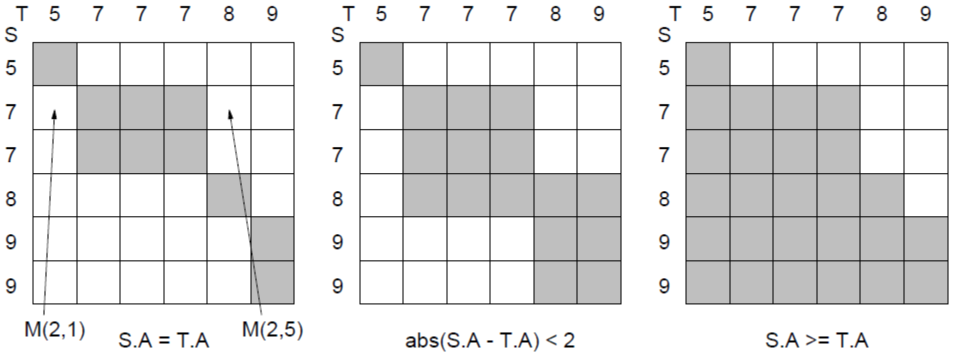


Optimization Goal: Minimal Job Completion time

- Job completion time depends on the slowest map and reduce functions
- Balancing the workloads of map functions is easy and thus we ignore them
- Balance the workloads of reduce functions as evenly as possible
  - Assume all reducers are similarly capable
- Processing time at reducer is approximately monotonic in input and output size
- Hence need to minimize max-reducer-input or max-reducer-output
- Join problem classification
  - Input-size dominated: minimize max-reducer-input
  - Output-size dominated: minimize max-reducer-output
  - Input-output balanced: minimize combination of both

Join Model

- Join-matrix M: M(i, j) = true, if and only if (s_i, t_j) in join result
- Cover each true-valued cell by exactly one reducer

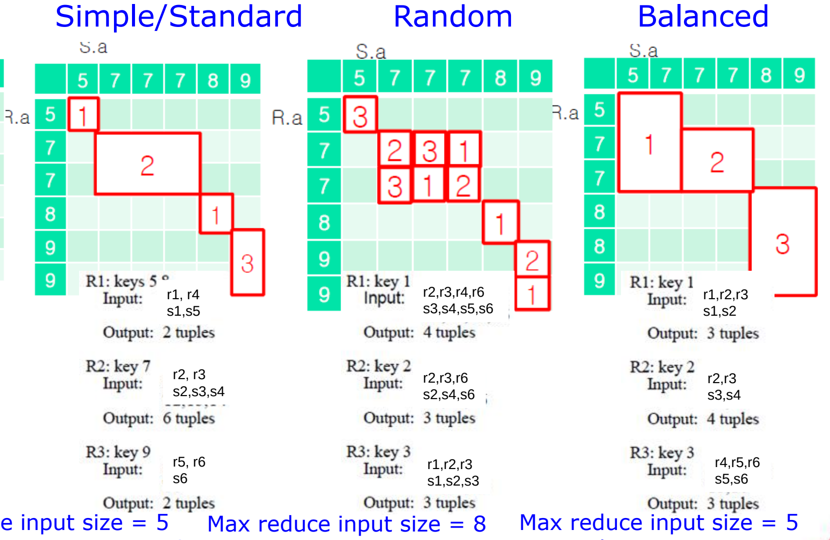


Reduce Allocations for Repartition Equi-joins

R	a
r_1	5
r_2	7
r_3	7
r_4	8
r_5	9
r_6	9

S	a
s_1	5
s_2	7
s_3	7
s_4	7
s_5	8
s_6	9

R.a = S.a



Simple/Standard	Random	Balanced
Max reduce input size = 5	Max reduce input size = 8	Max reduce input size = 5
Max reduce output size = 6	Max reduce output size = 4	Max reduce output size = 4

Comparison of Reduce Allocation Methods

- Simple allocation
  - Minimize the maximum input size of reduce functions
  - Output size may be skewed
- Random allocation
  - Minimize the maximum output size of reduce functions
  - Input size may be increased due to duplication
- Balanced allocation
  - Minimize both maximum input and output sizes

How to Balance Reduce Allocation

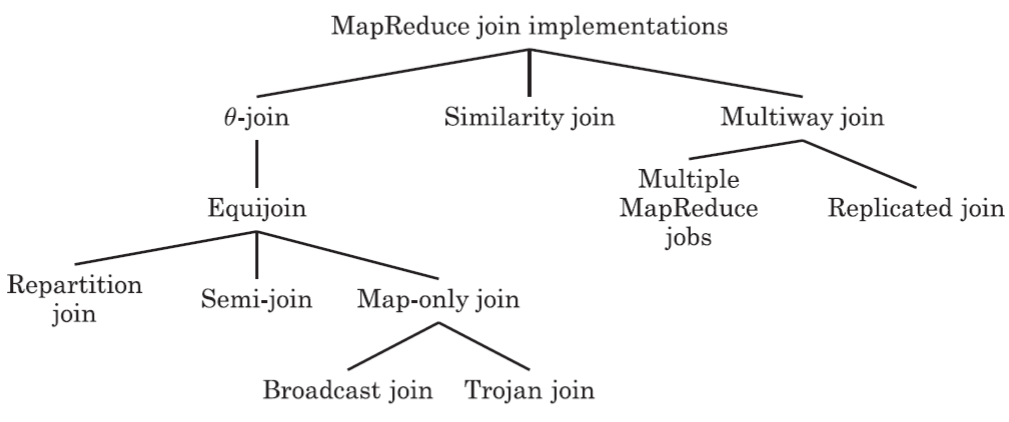
- Assume r is desired number of reduce functions
- Partition join-matrix M into r regions
- A map function sends each record in R and S to mapped regions
- A reduce function outputs all possible (r,s) pairs satisfying the join predicates in its value-list
- Propose M-Bucket-I algorithm [Okcan Riedewald: SIGMOD 2011]

25

Processing Relational Data: Summary

- MapReduce algorithms for processing relational data:
  - Group by, sorting, partitioning are handled automatically by shuffle/sort in MapReduce
  - Selection, projection, and other computations (e.g., aggregation), are performed either in mapper or reducer
- Complex operations require multiple MapReduce jobs
  - Example: top ten URLs in terms of average time spent
  - Opportunities for automatic optimization
- Multiple strategies for relational joins

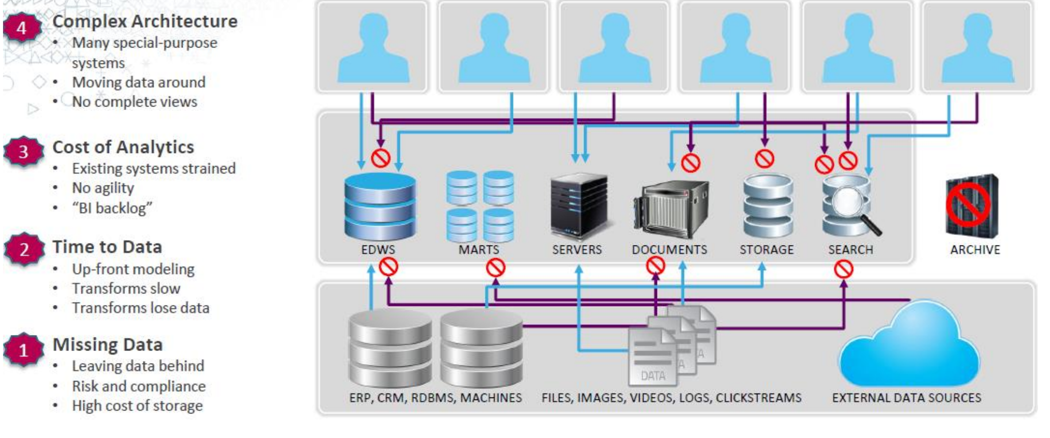
Join Implementations on MapReduce



Feng Li, Beng Chin Ooi, M. Tamer Özsu, and Sai Wu. 2014. Distributed data management using MapReduce. ACM Comput. Surv. 46, 3, January 2014

Evolving Roles for Relational Database and MapReduce

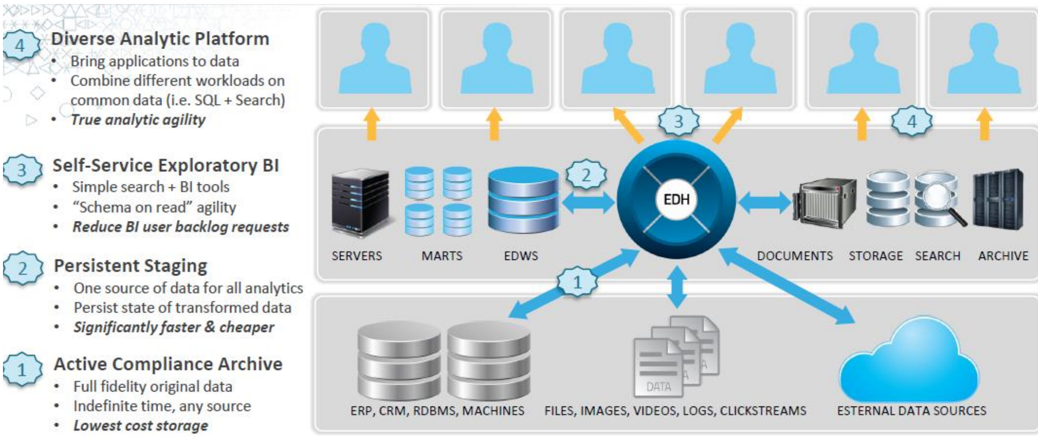
The Traditional Way: Bringing Data to Compute



Evolution from Apache Hadoop to the Enterprise Data Hub A. Awadallah Co-Founder & CTO of Cloudera SMDB 2014

27

The New Way: Bringing Compute to Data



Evolution from Apache Hadoop to the Enterprise Data Hub A. Awadallah Co-Founder & CTO of Cloudera SMDB 2014

Fall 2019

Need for High-Level Languages

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Analysts don't want to (or can't) write Java
- Solution: develop higher-level data processing languages
  - Hive: HQL is like SQL
  - Pig: Pig Latin is a bit like Perl

Hive and Pig

- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source
- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source
  - Roughly 1/3 of all Yahoo! internal jobs
- Common idea:
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language "compiles down" to Hadoop jobs

Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;

the     25848   62394
I       23031   8854
and     19671   38985
to      18038   13526
of      16700   34654
a       14170   8057
you     12702   2720
my      11297   4135
in      10797   12445
is      8882    6884
```

Source: Material drawn from Cloudera training VM

Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

(Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (.
(TOK_TABLE_OR_COL s) word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT
(TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT (TOK_SELEXPR (.
(TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR
(. (TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1)
(>= (. (TOK_TABLE_OR_COL k) freq) 1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (.
(TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```

(one or more of MapReduce jobs)

Hive: Behind the Scenes

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        s
          TableScan
            alias: s
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 0
                value expressions:
                      expr: freq
                      type: int
                      expr: word
                      type: string
        k
          TableScan
            alias: k
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 1
                value expressions:
                      expr: freq
                      type: int

      Reduce Operator Tree:
        Join Operator
          condition map:
               Inner Join 0 to 1
          condition expressions:
            0 {VALUE._col0} {VALUE._col1}
            1 {VALUE._col0}
          outputColumnNames: _col0, _col1, _col2
          Filter Operator
            predicate:
                expr: ((_col0 >= 1) and (_col2 >= 1))
                type: boolean
            Select Operator
              expressions:
                    expr: _col1
                    type: string
                    expr: _col0
                    type: int
                    expr: _col2
                    type: int
              outputColumnNames: _col0, _col1, _col2
              File Output Operator
                compressed: false
                GlobalTableId: 0
                table:
                    input format: org.apache.hadoop.mapred.SequenceFileInputFormat
                    output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat

  Stage: Stage-2
    Map Reduce
      Alias -> Map Operator Tree:
        hdfs://localhost:8022/tmp/hive-training/364214370/10002
            Reduce Output Operator
              key expressions:
                    expr: _col1
                    type: int
              sort order: -
              tag: -1
              value expressions:
                    expr: _col0
                    type: string
                    expr: _col1
                    type: int
                    expr: _col2
                    type: int
      Reduce Operator Tree:
        Extract
          Limit
            File Output Operator
              compressed: false
              GlobalTableId: 0
              table:
                  input format: org.apache.hadoop.mapred.TextInputFormat
                  output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

  Stage: Stage-0
    Fetch Operator
      limit: 10
```

Pig: Example

- Task: Find the top 10 most visited pages in each category

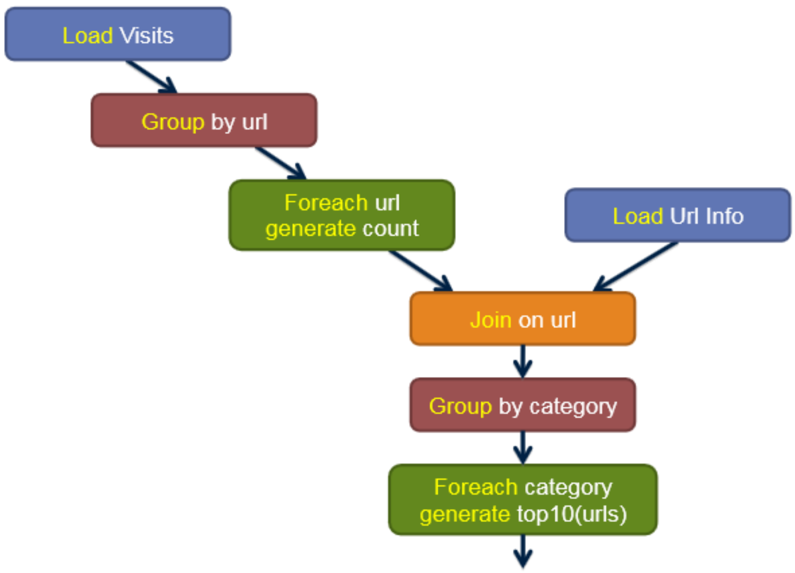
Visits

User	Url	Time
Amy	cnn.com	8:00
Amy	bbc.com	10:00
Amy	flickr.com	10:05
Fred	cnn.com	12:00

Url Info

Url	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9

Pig Query Plan



Fall 2019

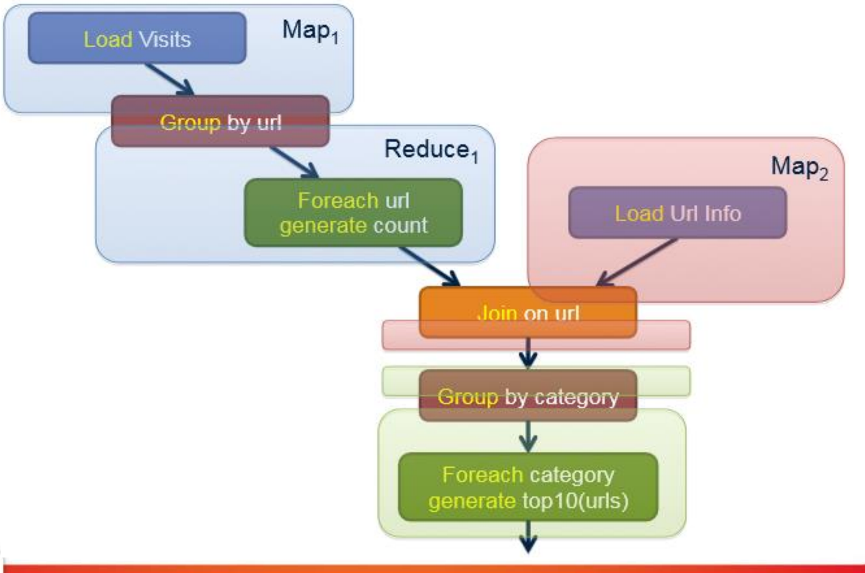
Pig Script

```
visits = load '/data/visits'  as (user,  url,  time);
gVisits =  group visits  by url;
visitCounts =  foreach gVisits generate url,  count(visits);
urlInfo =  load '/data/urlInfo'  as (url,  category,  pRank);
visitCounts =  join visitCounts by url,  urlInfo by url;
gCategories =  group visitCounts by category;
topUrls =  foreach gCategories generate top(visitCounts,10);

store  topUrls into  '/data/topUrls';
```


Fall 2019

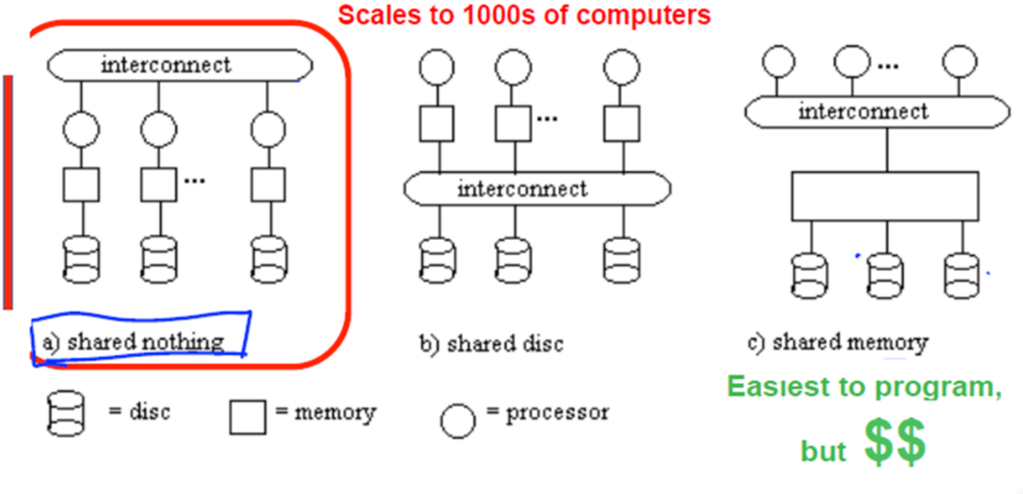
Pig Query Plan



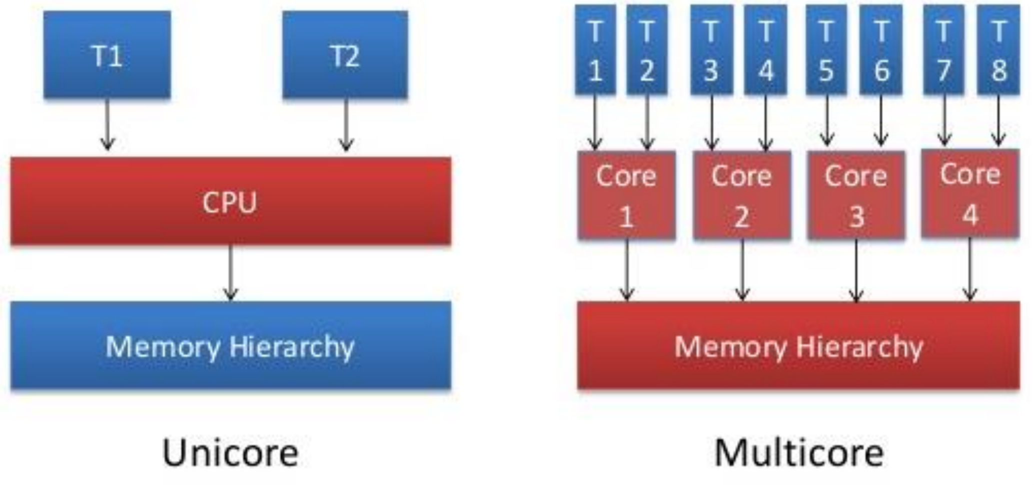
References

- CS9223 – Massive Data Analysis J. Freire & J. Simeon New York University Course 2013
- INFM 718G / CMSC 828G Data-Intensive Computing with MapReduce J. Lin University of Maryland 2013
- CS 6240: Parallel Data Processing in MapReduce Mirek Riedewald Northeastern University 2014
- Extreme Computing Stratis D. Viglas University of Edinburg 2014
- MapReduce Algorithms for Big Data Analysis Kyuseok Shim VLDB 2012 TUTORIAL

Taxonomy of Parallel Architectures



Unicore vs Multi-core Architectures



Positioning Big Data