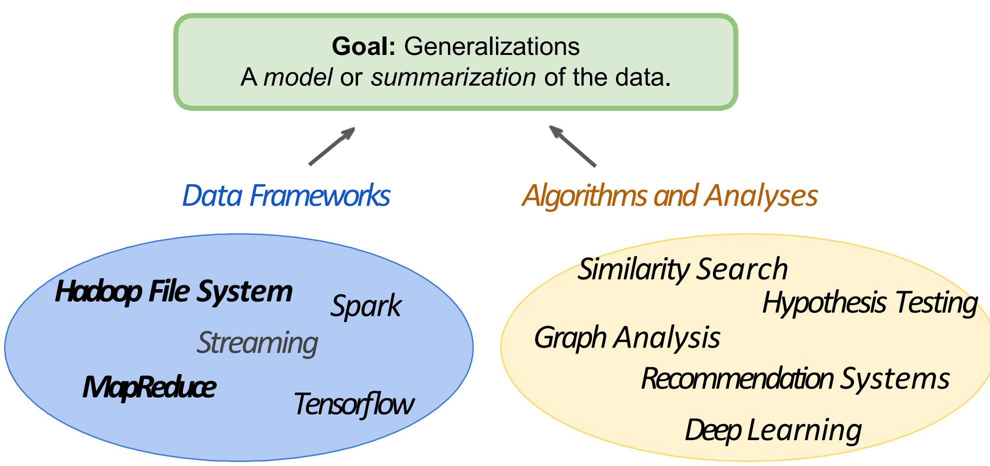
“Hadoop”:

A Distributed Architecture, FileSystem, & MapReduce

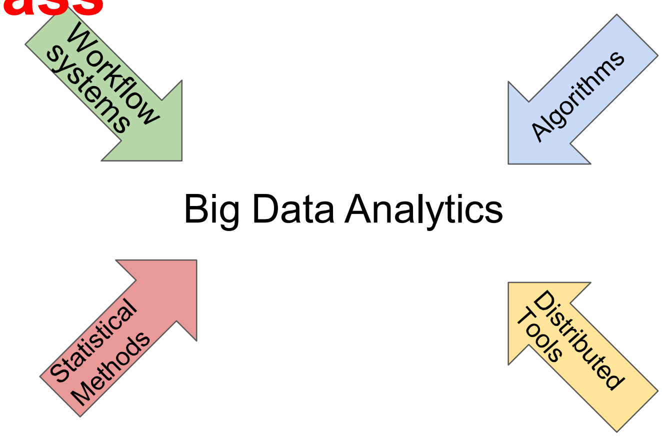
1

Big Data Analytics, The Class



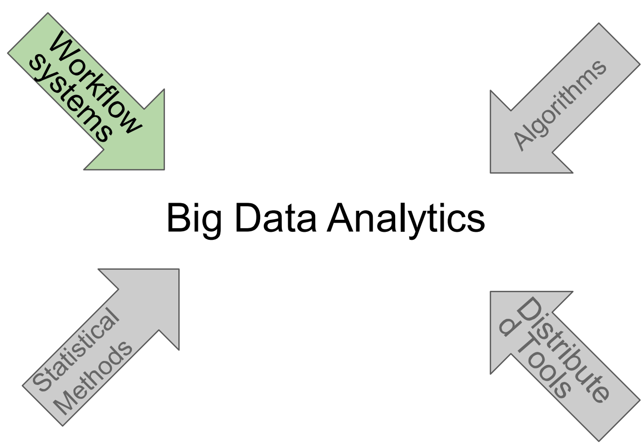
2

Big Data Analytics, The Class



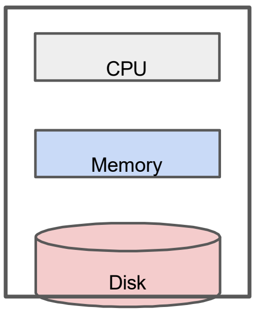
3

Big Data Analytics, The Class



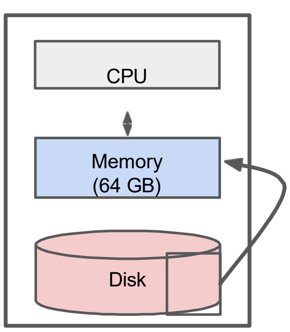
4

Classical Data Analytics



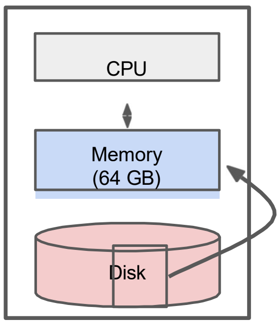
5

Classical Data Analytics



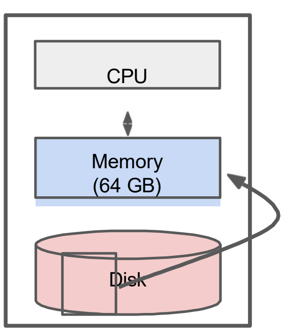
6

Classical Data Analytics



7

Classical Data Analytics

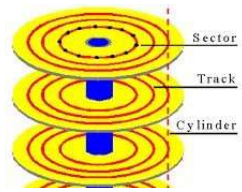


8

IO Bounded

Reading a word from disk versus main memory: 10^5 slower!

Reading many contiguously stored words
is faster per word, but fast modern disks
still only reach 150MB/s for sequential reads.

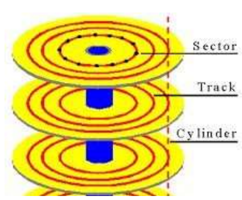


9

IO Bounded

Reading a word from disk versus main memory: 10^5 slower!

Reading many contiguously stored words
is faster per word, but fast modern disks
still only reach 150MB/s for sequential reads.



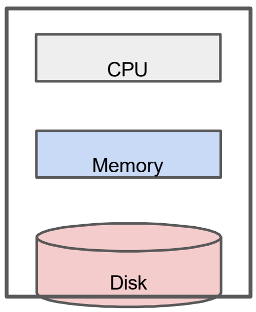
IO Bound: biggest performance bottleneck is reading / writing to disk.

starts around 100 GBs: ~10 minutes just to read

200 TBs: ~20,000 minutes = 13 days

10

Classical Big Data



Classical focus: efficient use of disk.
e.g. Apache Lucene / Solr

Classical limitation: Still bounded when needing to process all of a large file.

11

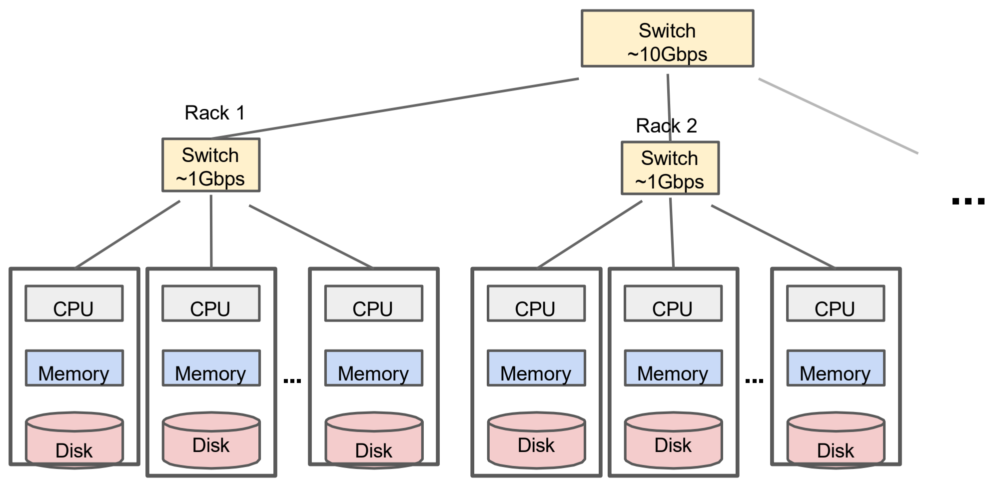
Classical Big Data

How to solve?

Classical limitation: Still bounded when needing to process all of a large file.

12

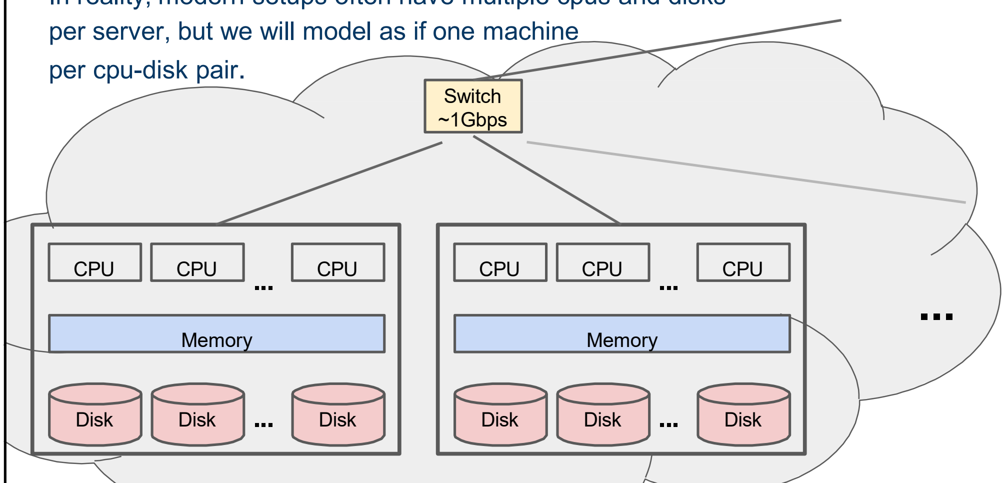
Distributed Architecture



13

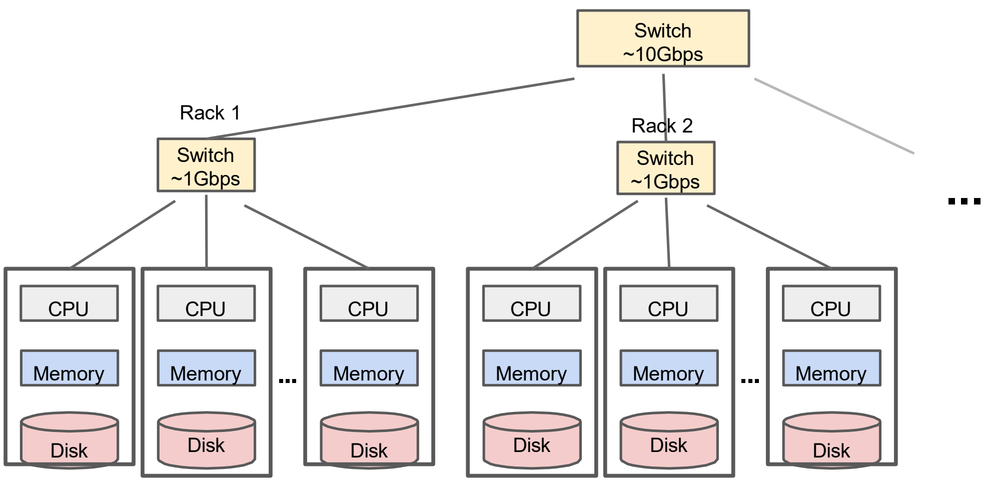
Distributed Architecture

In reality, modern setups often have multiple cpus and disks per server, but we will model as if one machine per cpu-disk pair.



14

Distributed Architecture (Cluster)



15

Distributed Architecture (Cluster)

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day

2. Network is a bottleneck
   Typically 1-10 Gb/s throughput

3. Traditional distributed programming is
   often ad-hoc and complicated

16

Distributed Architecture (Cluster)

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data
2. Network is a bottleneck
   Typically 1-10 Gb/s throughput
   Bring computation to nodes, rather than data to nodes.
3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed

17

Distributed Architecture (Cluster)

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data
2. Network is a bottleneck
   Typically 1-10 Gb/s throughput
   Bring computation to nodes, rather than data to nodes.
3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed

HDFS with MapReduce accomplishes all!

18

Distributed Filesystem

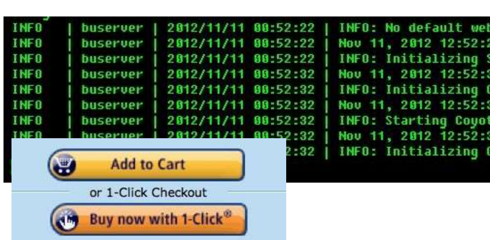
The effectiveness of MapReduce, Spark, and other distributed processing systems is in part simply due to use of a distributed filesystem!

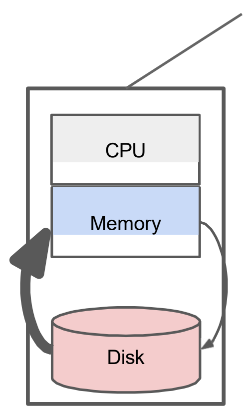
19

Distributed Filesystem

Characteristics for Big Data Tasks

- Large files (i.e. >100 GB to TBs)
- Reads are most common
- No need to update in place (append preferred)



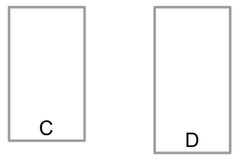


20

Distributed Filesystem

(e.g. Apache HadoopDFS, GoogleFS, EMRFS)

C, D: Two different files



https://opensource.com/life/14/8/intro-apache-hadoop-big-data

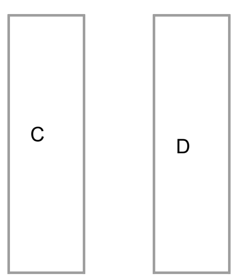
21

Distributed Filesystem

"Hadoop" was named after a toy elephant belonging to Doug Cutting's son. Cutting was one of Hadoop's creators.

(e.g. Apache **Hadoop**DFS, GoogleFS, EMRFS)

C, D: Two different files



https://opensource.com/life/14/8/intro-apache-hadoop-big-data

22

Distributed Filesystem

(e.g. Apache HadoopDFS, GoogleFS, EMRFS)

C, D: Two different files; break into chunks (or "partitions"):

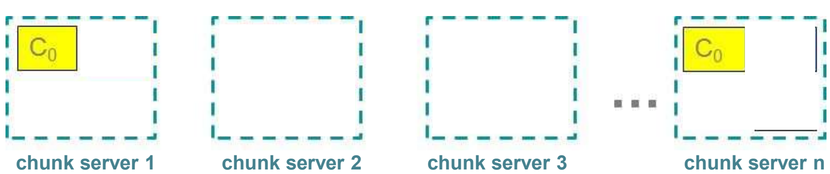
C_0	D_0
C_1	D_1
C_2	D_2
C_3	D_3
C_4	D_4
C_5	D_5

23

Distributed Filesystem

(e.g. Apache HadoopDFS, GoogleFS, EMRFS)

C, D: Two different files



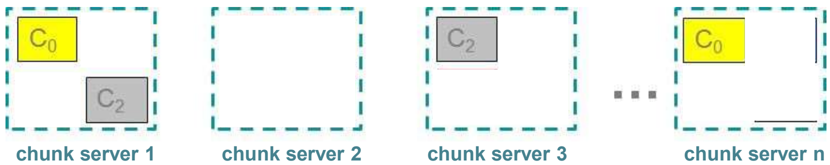
(Leskovec at al., 2014; http://www.mmds.org/)

24

Distributed Filesystem

(e.g. Apache HadoopDFS, GoogleFS, EMRFS)

C, D: Two different files



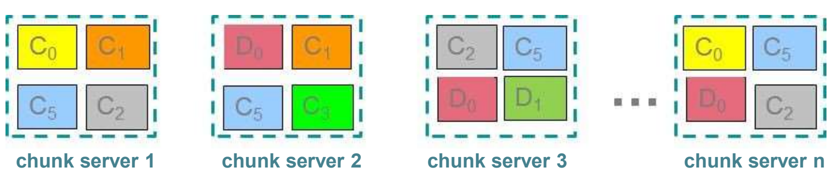
(Leskovec at al., 2014; http://www.mmds.org/)

25

Distributed Filesystem

(e.g. Apache HadoopDFS, GoogleFS, EMRFS)

C, D: Two different files



(Leskovec at al., 2014; http://www.mmds.org/)

26

Distributed Filesystem

Chunk servers (on Data Nodes)

- File is split into contiguous chunks
- Typically each chunk is 16-64MB
- Each chunk replicated (usually 2x or 3x)
- Try to keep replicas in different racks

(Leskovec at al., 2014; http://www.mmds.org/)

27

Components of a Distributed Filesystem

Chunk servers (on Data Nodes)

- File is split into contiguous chunks
- Typically each chunk is 16-64MB
- Each chunk replicated (usually 2x or 3x)
- Try to keep replicas in different racks

Name node (aka master node)

- Stores metadata about where files are stored
- Might be replicated or distributed across data nodes.

(Leskovec at al., 2014; http://www.mmds.org/)

28

Components of a Distributed Filesystem

Chunk servers (on Data Nodes)

File is split into contiguous chunks

Typically each chunk is 16-64MB

Each chunk replicated (usually 2x or 3x)

Try to keep replicas in different racks

Name node (aka master node)

Stores metadata about where files are stored

Might be replicated or distributed across data nodes.

Client library for file access

Talks to master to find chunk servers

Connects directly to chunk servers to access data

(Leskovec at al., 2014; http://www.mmds.org/)

29

Distributed Architecture (Cluster)

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data **(Distributed FS)** ✔
2. Network is a bottleneck
   Typically 1-10 Gb/s throughput
   Bring computation to nodes, rather than data to nodes.
3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed

30

What is MapReduce

noun.1* - A *style of programming

input chunks => map tasks | group_by keys | reduce tasks => output

"|" is the linux "pipe" symbol: passes stdout from first process to stdin of next.

31

What is MapReduce

noun.1* - A *style of programming

input chunks => map tasks | group_by keys | reduce tasks => output

"|" is the linux "pipe" symbol: passes stdout from first process to stdin of next.

E.g. counting words:

```
tokenize(document) | sort | uniq -c
```

32

What is MapReduce

noun.1 **- A *style of programming***

input chunks => map tasks | group_by keys | reduce tasks => output

"|" is the linux "pipe" symbol: passes stdout from first process to stdin of next.

E.g. counting words:

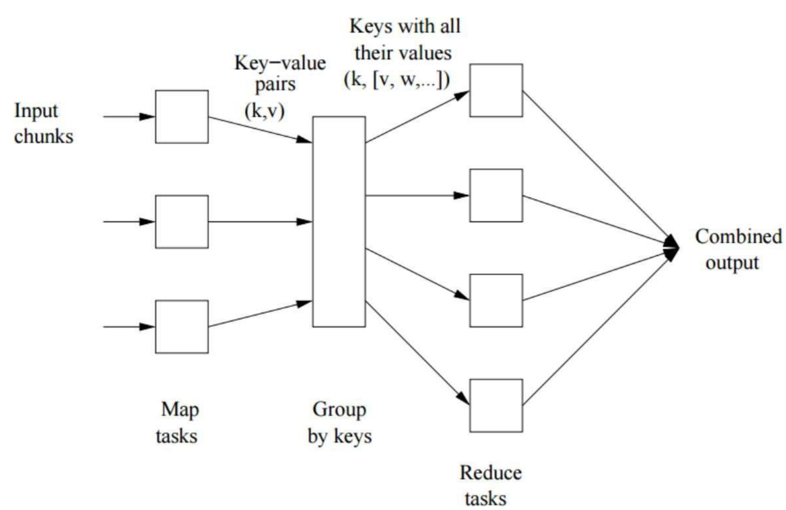
```
tokenize(document) | sort | uniq -c
```

noun.2 **- A *system* that distributes MapReduce style programs across a distributed file-system.**

(e.g. Google's internal "MapReduce" or apache.hadoop.mapreduce with hdfs)

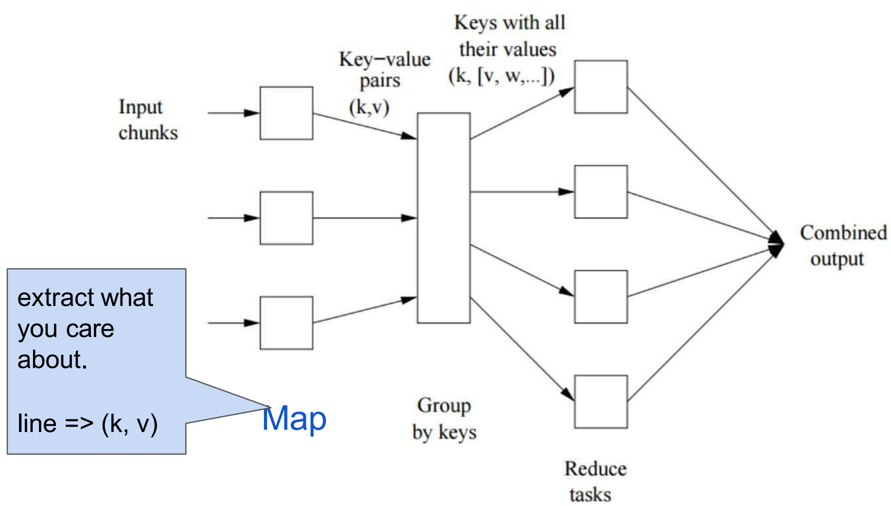
33

What is MapReduce



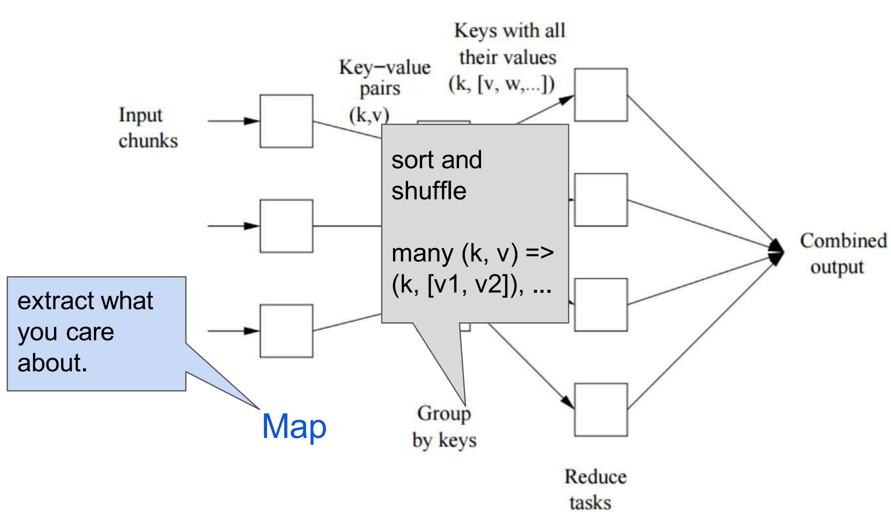
34

What is MapReduce



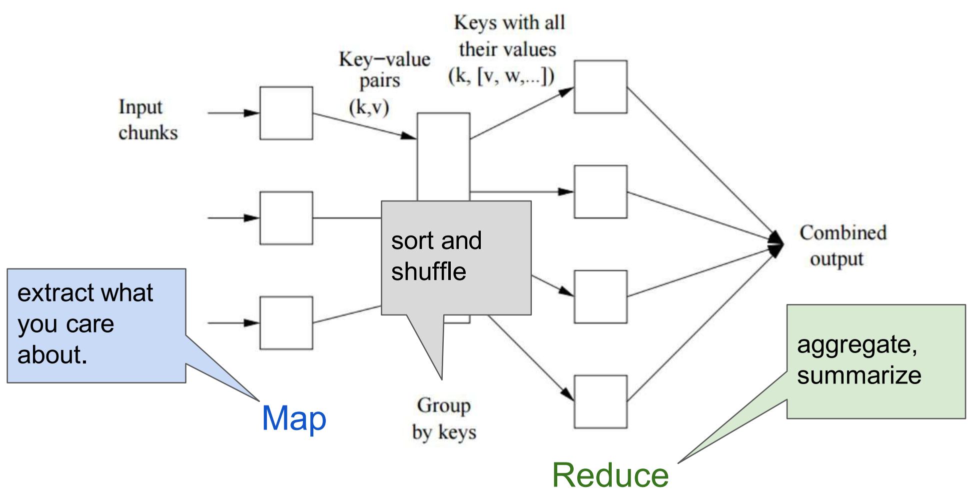
35

What is MapReduce



36

What is MapReduce



37

What is MapReduce

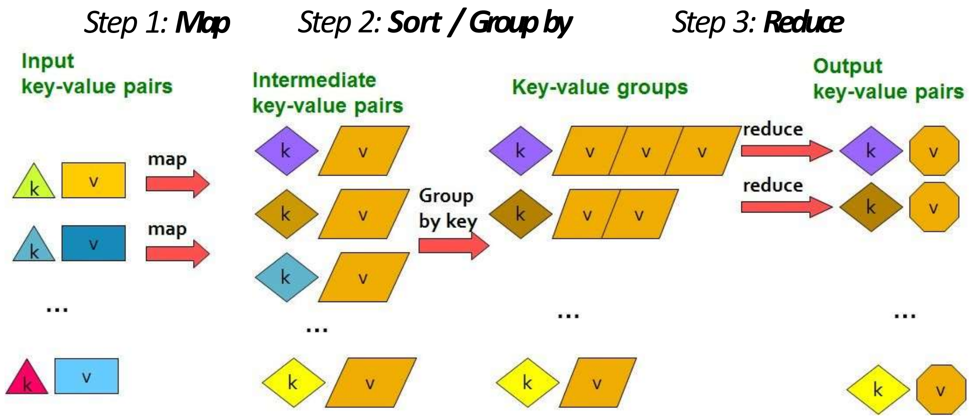
Easy as 1, 2, 3!

*Step 1: **Map*** *Step 2: **Sort / Group by*** *Step 3: **Reduce***

38

What is MapReduce

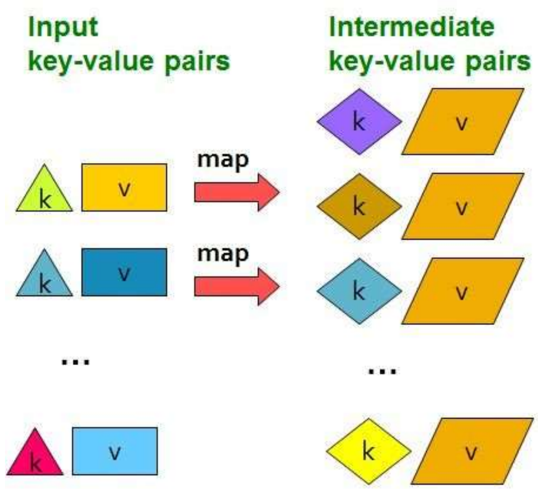
Easy as 1, 2, 3!



(Leskovec at al., 2014; http://www.mmds.org/)

39

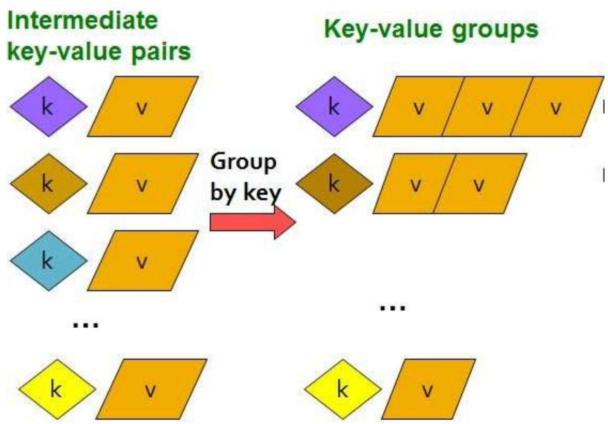
(1) The *Map* Step



(Leskovec at al., 2014; http://www.mmds.org/)

40

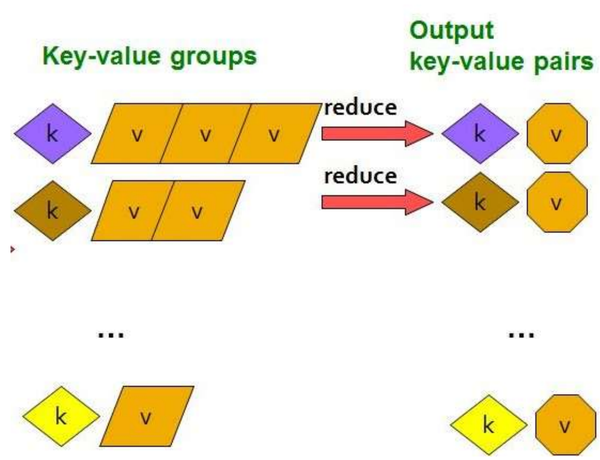
(2) The *Sort / Group-by* Step



(Leskovec at al., 2014; http://www.mmds.org/)

41

(3) The *Reduce* Step



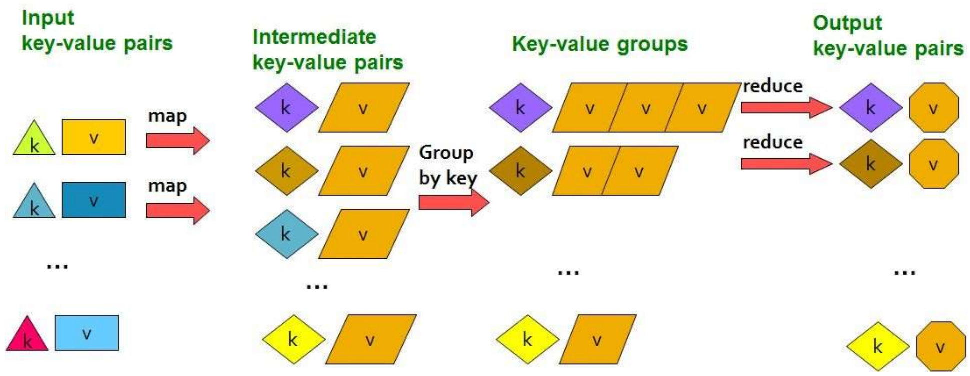
(Leskovec at al., 2014; http://www.mmds.org/)

42

What is MapReduce

Easy as 1, 2, 3!

Step 1: ***Map*** *Step 2:* ***Sort / Group by*** *Step 3:* ***Reduce***



(Leskovec at al., 2014; http://www.mmds.org/)

43

What is MapReduce

Map: (k,v) -> (k', v')*
(Written by programmer)

Group by key: (k_1', v_1'), (k_2', v_2'), ... -> (k_1', (v_1', v', …),
(system handles) (k_2', (v_1', v', …), …

Reduce: (k', (v_1', v', …)) -> (k', v'')*
(Written by programmer)

44

Example: Word Count

```
tokenize(document) | sort | uniq -c
```

45

Example: Word Count

```
tokenize(document) | sort | uniq -c
```

Map: extract what you care about.

sort and shuffle

Reduce: aggregate, summarize

46

Example: Word Count

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ....

Big document

(Leskovec at al., 2014; http://www.mmds.org/)

47

Provided by the programmer

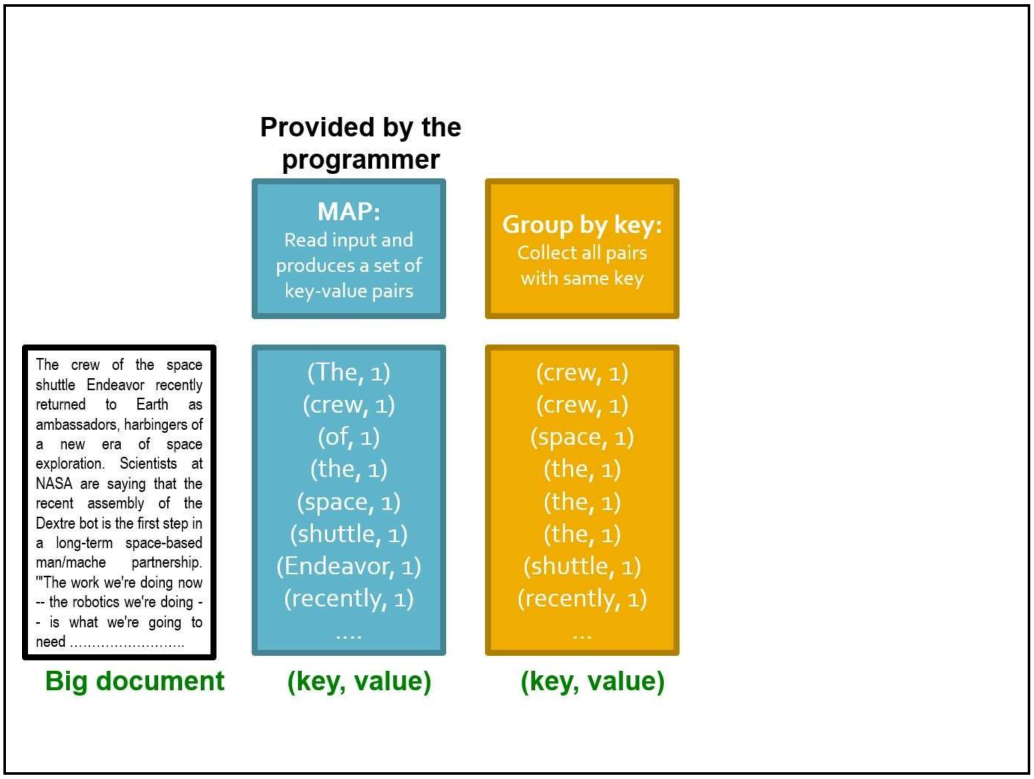
MAP:
Read input and produces a set of key-value pairs

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ....

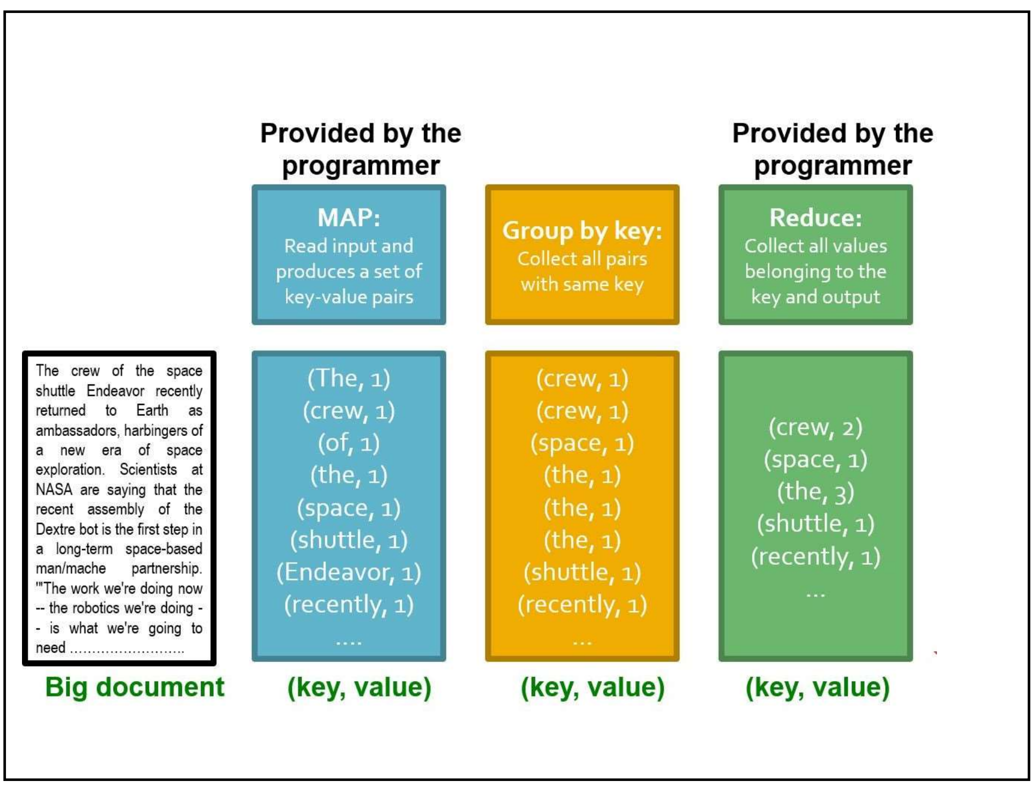
(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....

Big document **(key, value)**

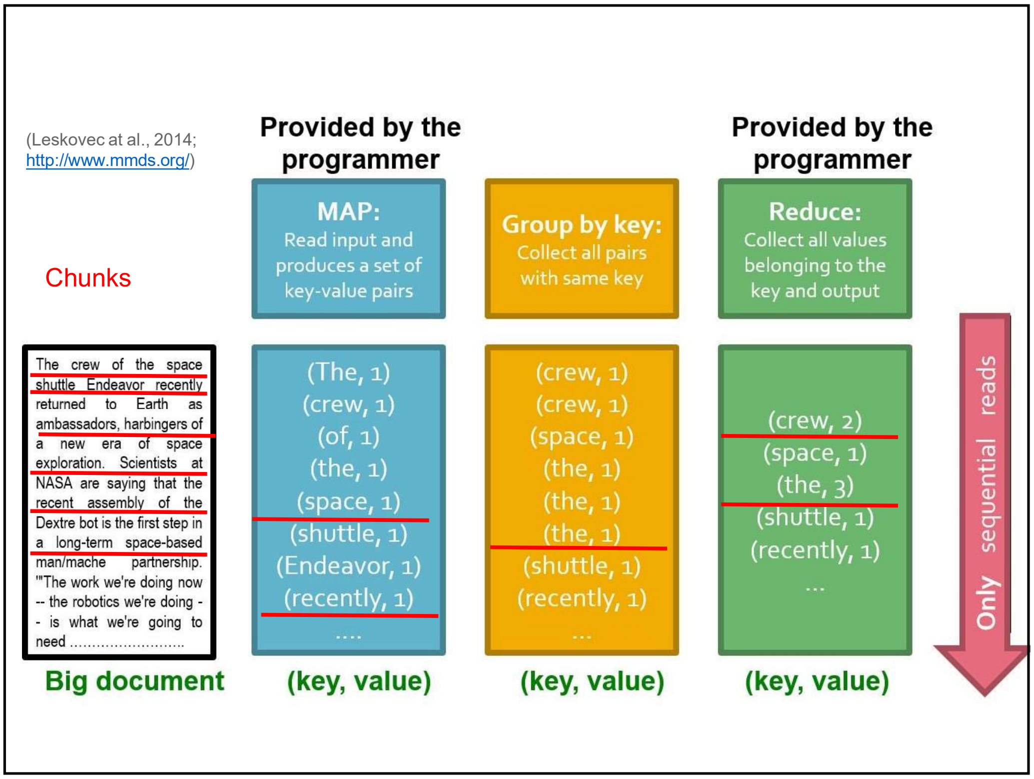
48


Provided by the programmer
MAP:
Read input and produces a set of key-value pairs
Group by key:
Collect all pairs with same key
The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ....
(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....
(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
...
Big document
(key, value)
(key, value)

49


Provided by the programmer
Provided by the programmer
MAP:
Read input and produces a set of key-value pairs
Group by key:
Collect all pairs with same key
Reduce:
Collect all values belonging to the key and output
The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ....
(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....
(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
...
(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
...
Big document
(key, value)
(key, value)
(key, value)

50



51

Example: Word Count

```python
@abstractmethod
def map(k, v):
    pass


@abstractmethod
def reduce(k, vs):
    pass
```

52

Example: Word Count (v1)

```
def map(k, v):
    for w in tokenize(v):
        yield (w,1)



def reduce(k, vs):
    return len(vs)
```

53

Example: Word Count (v1)

```
def map(k, v):
    for w in tokenize(v):
        yield (w,1)



def reduce(k, vs):
    return len(vs)
```

```
def tokenize(s):
    #simple version
    return s.split(‘ ‘)
```

54

Example: Word Count (v2)

```
def map(k, v):
    counts = dict()
    for w in tokenize(v):
```

counts each word within the chunk
(try/except is faster than
"if w in counts")

55

Example: Word Count (v2)

```
def map(k, v):
    counts = dict()
    for w in tokenize(v):
        try:
            counts[w] += 1
        except KeyError:
            counts[w] = 1
    for item in counts.iteritems():
        yield item
```

counts each word within the chunk
(try/except is faster than
"if w in counts")

56

Example: Word Count (v2)

```
def map(k, v):
    counts = dict()
    for w in tokenize(v):
        try:
            counts[w] += 1
        except KeyError:
            counts[w] = 1
    for item in counts.iteritems():
        yield item
```

counts each word within the chunk (try/except is faster than "if w in counts")

```
def reduce(k, vs):
    return (k, sum(vs) )
```

sum of counts from different chunks

57

Distributed Architecture (Cluster)

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data **(Distributed FS)** ✔
2. Network is a bottleneck
   Typically 1-10 Gb/s throughput
   Bring computation to nodes, rather than data to nodes.
3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed

58

Distributed Architecture (Cluster)

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data **(Distributed FS)**
2. Network is a bottleneck
   Typically 1-10 Gb/s throughput
   Bring computation to nodes, rather than data to nodes. **(Sort and Shuffle)**
3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed

59

Distributed Architecture (Cluster)

Challenges for IO Cluster Computing

1. Nodes fail
   1 in 1000 nodes fail a day
   Duplicate Data **(Distributed FS)**
2. Network is a bottleneck
   Typically 1-10 Gb/s throughput
   Bring computation to nodes, rather than data to nodes. **(Sort and Shuffle)**
3. Traditional distributed programming is often ad-hoc and complicated
   Stipulate a programming system that can easily be distributed **(Simply define a map and reduce)**

60

Example: Relational Algebra

Select

Project

Union, Intersection, Difference

Natural Join

Grouping

61

Example: Relational Algebra

Select

Project

Union, Intersection, Difference

Natural Join

Grouping

62

Example: Relational Algebra

Select

$R(A_1,A_2,A_3,...)$, Relation R, Attributes A_*

return only those attribute tuples where condition C is true

63

Example: Relational Algebra

Select

$R(A_1,A_2,A_3,...)$, Relation R, Attributes A_*

return only those attribute tuples where condition C is true

```
def map(k, v): #v is list of attribute tuples: [(...,), (...,), ...]
    r = []
    for t in v:
        if t satisfies C:
            r += [(t, t)]
    return r
```

64

Example: Relational Algebra

Select

R(A_1*,*A_2*,*A_3*,...)*, Relation *R,* Attributes *A*$_*$

return only those attribute tuples where condition *C* is true

```
def map(k, v): #v is list of attribute tuples: [(...,), (...,), ...]
    r = []
    for t in v:
        if t satisfies C:
            r += [(t, t)]
    return r
                def reduce(k, vs):
                  r = []
                  for each v in vs:
                    r += [(k, v)]
                  return r
```

65

Example: Relational Algebra

Select

R(A_1*,*A_2*,*A_3*,...)*, Relation *R,* Attributes *A*$_*$

return only those attribute tuples where condition *C* is true

```
def map(k, v): #v is list of attribute tuples
    for t in v:
        if t satisfies C:
            yield (t, t)

def reduce(k, vs):
    For each v in vs:
        yield  (k, v)
```

66

Example: Relational Algebra

Natural Join

Given R_1 and R_2 return R_{join}

-- union of all pairs of tuples that match given attributes.

```
def map(k, v): #k \in {R1, R2}, v is (A, B) for R1, (B, C) for
               R2 #B are matched attributes
```

67

Example: Relational Algebra

Natural Join

Given R_1 and R_2 return R_{join}

-- union of all pairs of tuples that match given attributes.

```
def map(k, v): #k \in {R1, R2}, v is (A, B) for R1, (B, C) for
               R2 #B are matched attributes
    if k=='R1':
        (a, b) = v
        return (b,('R1',a))
    if k=='R2':
        (b,c) = v
        return (b,('R2',c))
```

68

Example: Relational Algebra

Natural Join

Given R_1 and R_2 return R_{join}

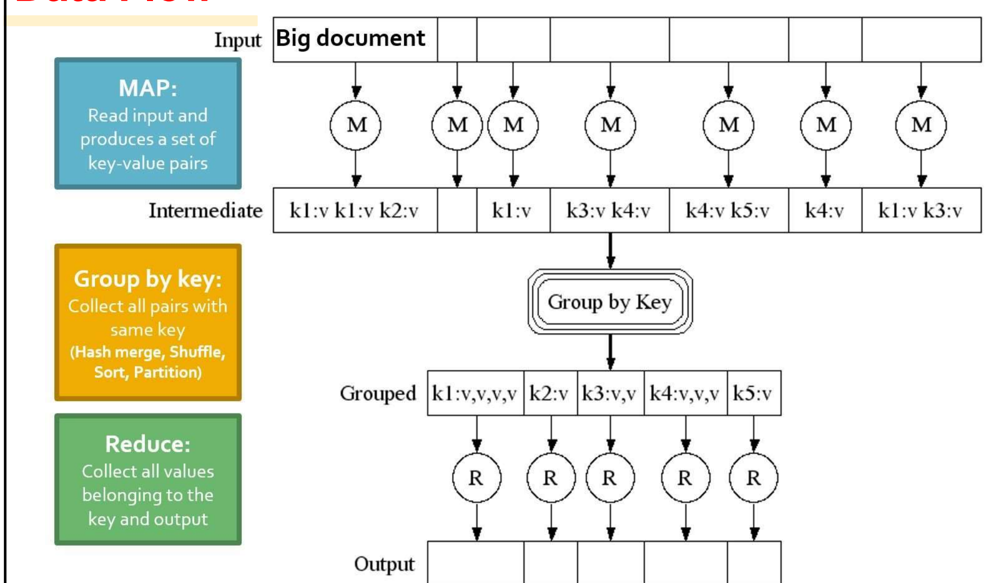
-- union of all pairs of tuples that match given attributes.

```
def map(k, v): #k \in {R1, R2}, v is (A, B) for R1,  (B, C) for R2
               #B are matched attributes
    if k=='R1':
        (a, b) = v
        return (b,('R1',a))
    if k=='R2':
        (b,c) = v
        return (b,('R2',c))
```

```
def reduce(k, vs):
    r1, r2, rjn = [], [], []
    for (s, x) in vs: #separate rs
      if s == 'R1': r1.append(x)
      else: r2.append(x)
    for a in r1: #join as tuple
      for each c in r2:
        rjn += ('Rjoin', (a, k, c)) #k is b
    return rjn
```

69

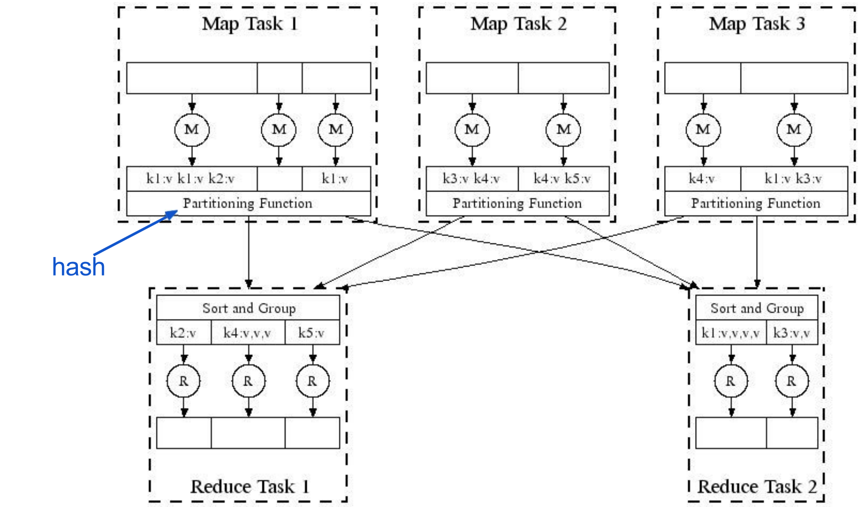
Data Flow



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 21

70

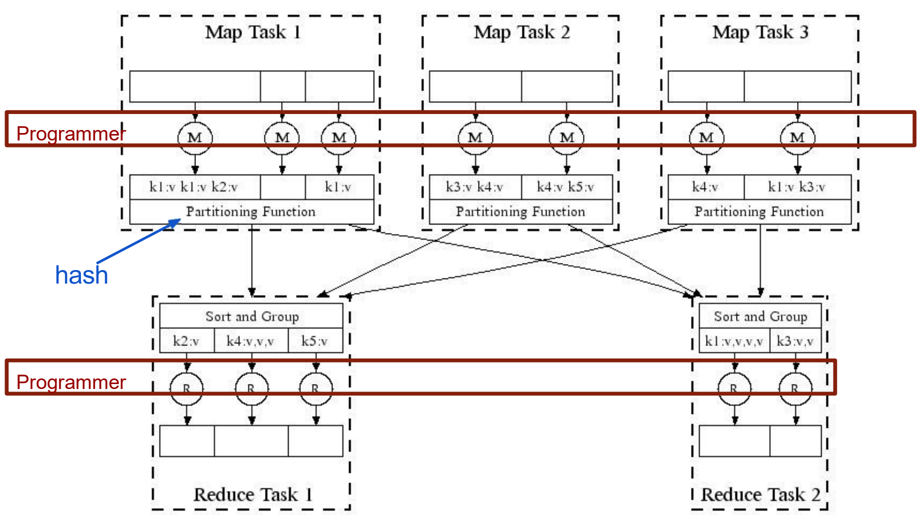
Data Flow



(Leskovec at al., 2014; http://www.mmds.org/)

71

Data Flow



(Leskovec at al., 2014; http://www.mmds.org/)

72

Data Flow

DFS ⇨ Map ⇨ Map's Local FS ⇨ Reduce ⇨ DFS

73

Data Flow

MapReduce system handles:

- Partitioning
- Scheduling map / reducer execution
- Group by key

- Restarts from node failures
- Inter-machine communication

74

Data Flow

DFS → MapReduce → DFS

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates

75

Data Flow

DFS → MapReduce → DFS

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates
  - Task status: idle, in-progress, complete
  - Receives location of intermediate results and schedules with reducer
  - Checks nodes for failures and restarts when necessary
    - All map tasks on nodes must be completely restarted
    - Reduce tasks can pickup with reduce task failed

76

Data Flow

DFS ⇨ MapReduce ⇨ DFS

- Schedule map tasks near physical storage of chunk
- Intermediate results stored locally
- Master / Name Node coordinates
    - Task status: idle, in-progress, complete
    - Receives location of intermediate results and schedules with reducer
    - Checks nodes for failures and restarts when necessary
        - All map tasks on nodes must be completely restarted
        - Reduce tasks can pickup with reduce task failed

DFS ⇨ MapReduce ⇨ DFS ⇨ MapReduce ⇨ DFS

77

Data Flow

Skew: The degree to which certain tasks end up taking much longer than others.

Handled with:

- More reducers than reduce tasks
- More reduce tasks than nodes

78

Data Flow

Key Question: *How many Map and Reduce jobs?*

M: map tasks, *R:* reducer tasks

79

Data Flow

Key Question: *How many Map and Reduce jobs?*

M: map tasks, *R:* reducer tasks

Answer: **1)** If possible, one chunk per map task, and

2) M >> |nodes| ≈≈ |cores|

(better handling of node failures, better load balancing)

3) R <= M

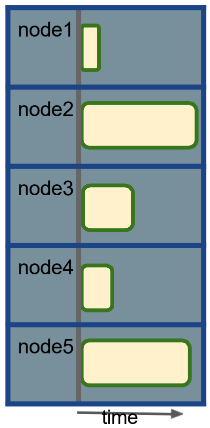
(reduces number of parts stored in DFS)

80

Data Flow

Reduce Task

version 1: few reduce tasks
(same number of reduce tasks as nodes)



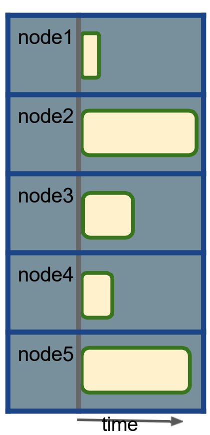
Reduce tasks represented by
time to complete task
(some tasks take much longer)

81

Data Flow

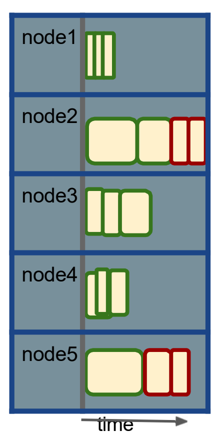
Reduce Task

version 1: few reduce tasks
(same number of reduce tasks as nodes)



Reduce tasks represented by
time to complete task
(some tasks take much longer)

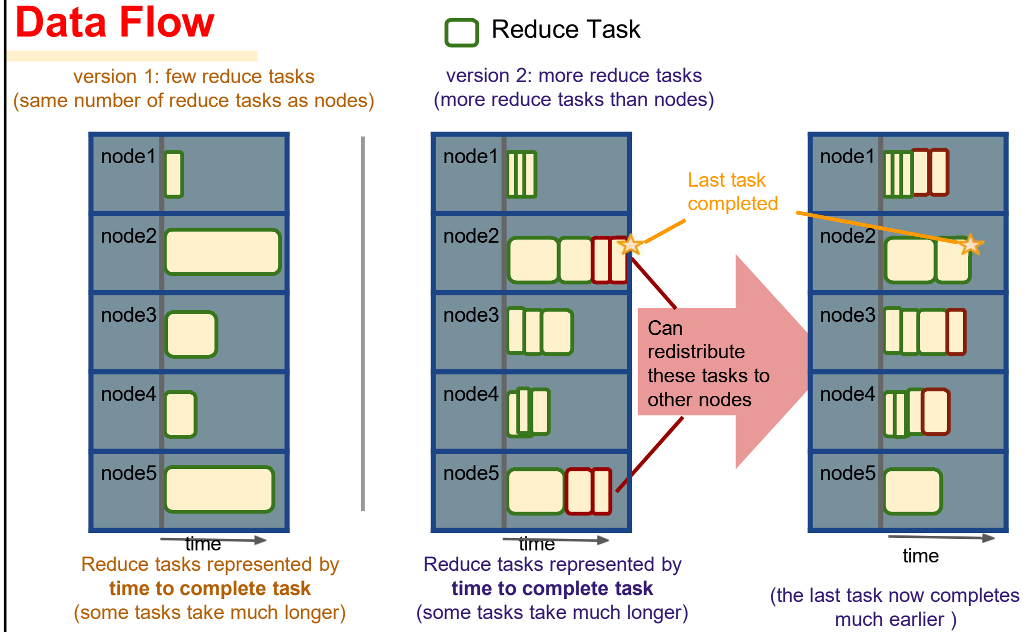
version 2: more reduce tasks
(more reduce tasks than nodes)



Reduce tasks represented by
time to complete task
(some tasks take much longer)

82

Data Flow



83

Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

(2) Communication: Moving (key, value) pairs

84

Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

(2) Communication: Moving (key, value) pairs

Ultimate Goal: wall-clock Time.

85

Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

- Mappers and reducers often single pass O(n) within node
- System: sort the keys is usually most expensive
- Even if map executes on same node, disk read usually dominates
- In any case, can add more nodes

(2) Communication: Moving (key, value) pairs

Ultimate Goal: wall-clock Time.

86

Communication Cost Model

How to assess performance?

(1) Computation: Map + Reduce + System Tasks

(2) Communication: Moving key, value pairs

Often dominates computation.

- Connection speeds: 1-10 giga**bits** per sec;

Ultimate Goal: wall-clock Time

HD read: 50-150 giga**bytes** per sec

- Even reading from disk to memory typically takes longer than operating on the data.

87

Communication Cost Model

How to assess performance?

Communication Cost = input size + (sum of size of all map-to-reducer files)

(2) Communication: Moving key, value pairs

Often dominates computation.

- Connection speeds: 1-10 giga**bits** per sec;

Ultimate Goal: wall-clock Time

HD read: 50-150 giga**bytes** per sec

- Even reading from disk to memory typically takes longer than operating on the data.

88

Communication Cost Model

How to assess performance?

Communication Cost =	input size + (sum of size of all map-to-reducer files)

(2) Communication: Moving key, value pairs

Often dominates computation.

- Connection speeds: 1-10 giga**bits** per sec;
  HD read: 50-150 giga**bytes** per sec
- Even reading from disk to memory typically takes longer than operating on the data.
- Output from reducer ignored because it's either small (finished summarizing data) or being passed to another mapreduce job.

Ultimate Goal: wall-clock Time

89

Communication Cost: Natural Join

R, S: Relations (Tables) *R(A, B)* $\bowtie$ *S(B, C)*

Communication Cost =	input size + (sum of size of all map-to-reducer files)

DFS ⇨ Map ⇨ LocalFS ⇨ Network ⇨ Reduce ⇨ DFS ⇨ ?

90

Communication Cost: Natural Join

R, S: Relations (Tables) $R(A, B) \bowtie S(B, C)$

Communication Cost = input size + (sum of size of all map-to-reducer files)

```
def map(k, v):
    if k=="R1":
        (a, b) = v
        yield (b,(R1,a))
    if k=="R2":
        (b,c) = v
        yield (b,(R2,c))
```

```
def reduce(k, vs):
    r1, r2 = [], []
    for (rel, x) in vs: #separate rs
        if rel == 'R': r1.append(x)
        else: r2.append(x)
    for a in r1: #join as tuple
        for each c in r2:
            yield (Rjoin', (a, k, c)) #k is
```

91

Communication Cost: Natural Join

R, S: Relations (Tables) $R(A, B) \bowtie S(B, C)$

Communication Cost = input size + (sum of size of all map-to-reducer files)

= |R1| + |R2| + (|R1| + |R2|)

= $O(|R1| + |R2|)$

```
def map(k, v):
    if k=="R1":
        (a, b) = v
        yield (b,(R1,a))
    if k=="R2":
        (b,c) = v
        yield (b,(R2,c))
```

```
def reduce(k, vs):
    r1, r2 = [], []
    for (rel, x) in vs: #separate rs
        if rel == 'R': r1.append(x)
        else: r2.append(x)
    for a in r1: #join as tuple
        for each c in r2:
            yield (Rjoin', (a, k, c)) #k is
```

92

MapReduce: Final Considerations

- Performance Refinements:
  - Combiners (like word count version 2 but done via reduce)
    - Run reduce right after map from same node before passing to reduce (MapTask can execute)
    - Reduces communication cost

    Requires commutative and associative reducer function.

93

MapReduce: Final Considerations

- Performance Refinements:
  - Combiners (like word count version 2 but done via reduce)
    - Run reduce right after map from same node before passing to reduce (MapTask can execute)
    - Reduces communication cost
  - Backup tasks (aka speculative tasks)
    - Schedule multiple copies of tasks when close to the end to mitigate certain nodes running slow.
  - Override partition hash function to organize data

    E.g. instead of `hash(url)` use `hash(hostname(url))`

94