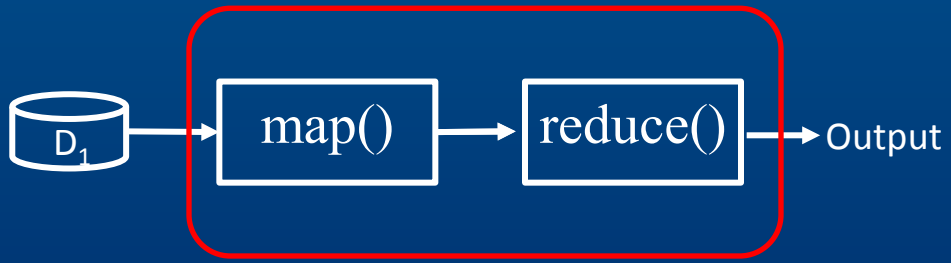
Introduction to Map/Reduce

Examples and Principles

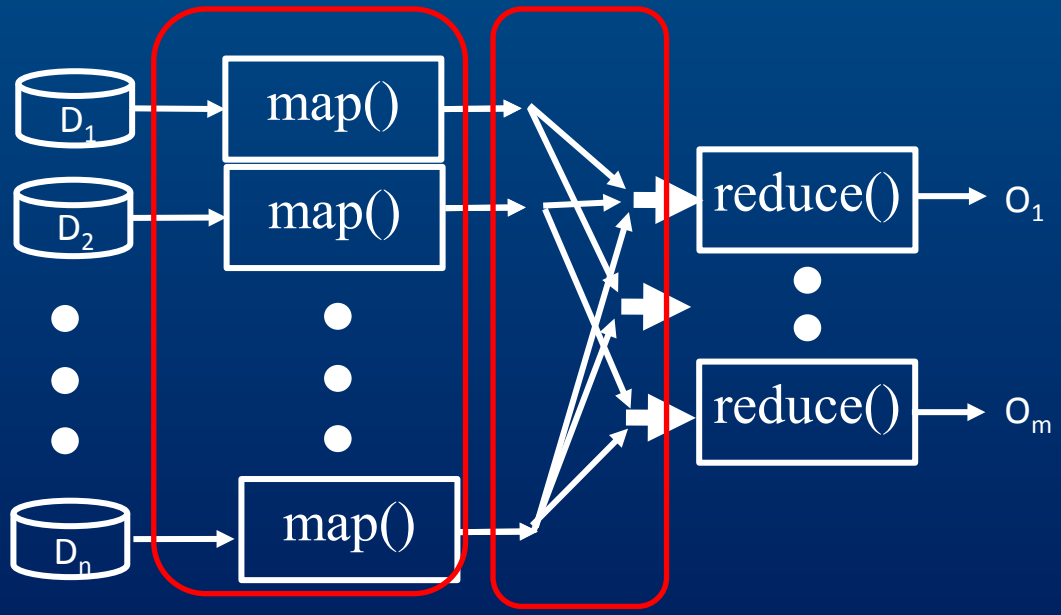
Recall the framework:

- User defines <key,value>, mapper, and reducer



Recall the framework:

- Hadoop handles the logistics



Hadoop Rule of Thumb

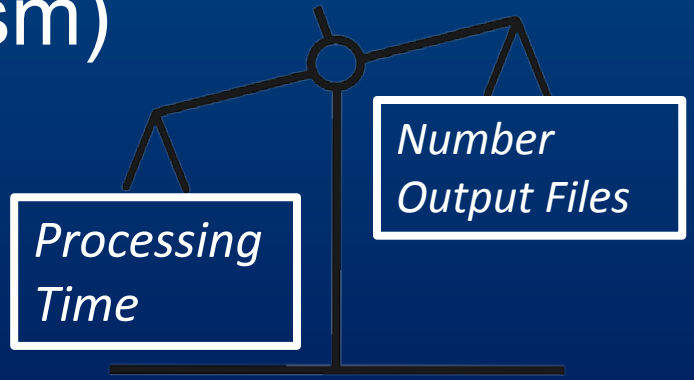
- 1 mapper per data split (typically)

Hadoop Rule of Thumb

- 1 mapper per data split (typically)
- 1 reducer per computer core (best parallelism)

Hadoop Rule of Thumb

- 1 mapper per data split (typically)
- 1 reducer per computer core (best parallelism)

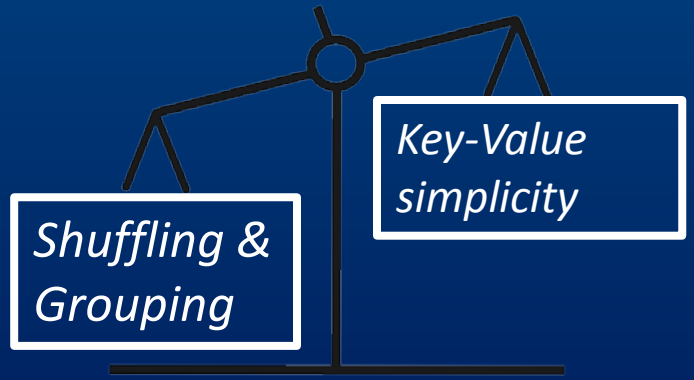


Wordcount Strategy

- Let <word, 1> be the <key,value>
- Simple mapper & reducer
- Hadoop did the hard work of shuffling & grouping

Good key-value properties

- Simple
- Enables reducers to get correct output



Good Task Decomposition:

Mappers: simple and separable

R*educers:* easy consolidation

Example:
Trending Wordcount

Trending Wordcount

- Twitter Data: date, message, location, [other metadata]

Trending Wordcount

- Twitter Data: date, message, location, [other metadata]

Task 1 Get word count by day

Task 2 Get total word count

Trending Wordcount

Task 1: get word count by day

Trending Wordcount

Task 1: get word count by day

Design: *Use composite key*

Map/Reduce: <date word,count>

Trending Wordcount

Task 2: get total word count

Trending Wordcount

Task 2: get total word count

Easy way:

re-use previous wordcount

Trending Wordcount

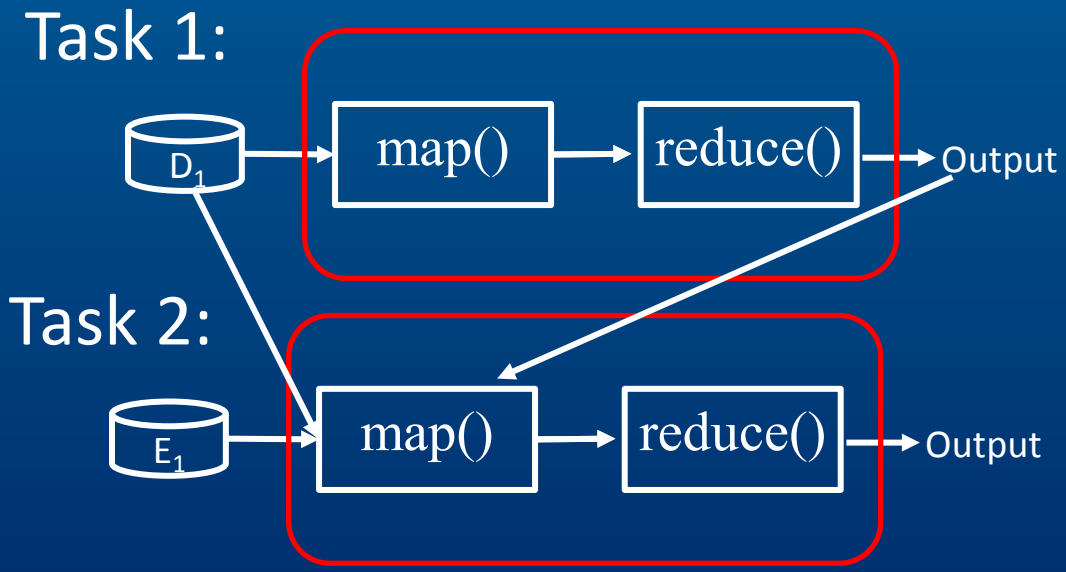
Task 2: get total word count

Alternatively:

use Task 1 output

(it's partially aggregated)

Cascading Map/Reduce



Task 3 …

Example:
Joining Data

Joining Data

- Task: combine datasets by key
  - A standard data management function

Joining Data

- Task: combine datasets by key
  - A standard data management function
  - In pseudo SQL

    Select * from table A, table B, where A.key=B.key

Joining Data

- Task: combine datasets by key
  - A standard data management function
  - In pseudo SQL

    Select * from table A, table B, where A.key=B.key
  - Joins can be inner, left or right outer

Joining Data

- Task: given two wordcount datasets

Joining Data

- Task: given two wordcount datasets

File A: <word, total-count>

```
able ,   5
actor ,   18
burger ,  25
.
.
.
```

Joining Data

- Task: given two wordcount datasets

File A: <word, total-count>

able , 5
actor , 18
burger , 25
.
.
.

File B: <date word, day-count>

Jan-16 able , 2
Feb-22 actor , 15
May-03 actor , 3
Jul-4 burger, 20
.
.
.

Joining Data

- Task: combine by word

File A: <word, total-count>

able , 5
actor , 18
burger , 25
.
.
.

File B: <date word, day-count>

Jan-16 able , 2
Feb-22 actor , 15
May-03 actor , 3
Jul-04 burger, 20
.
.
.

Joining Data

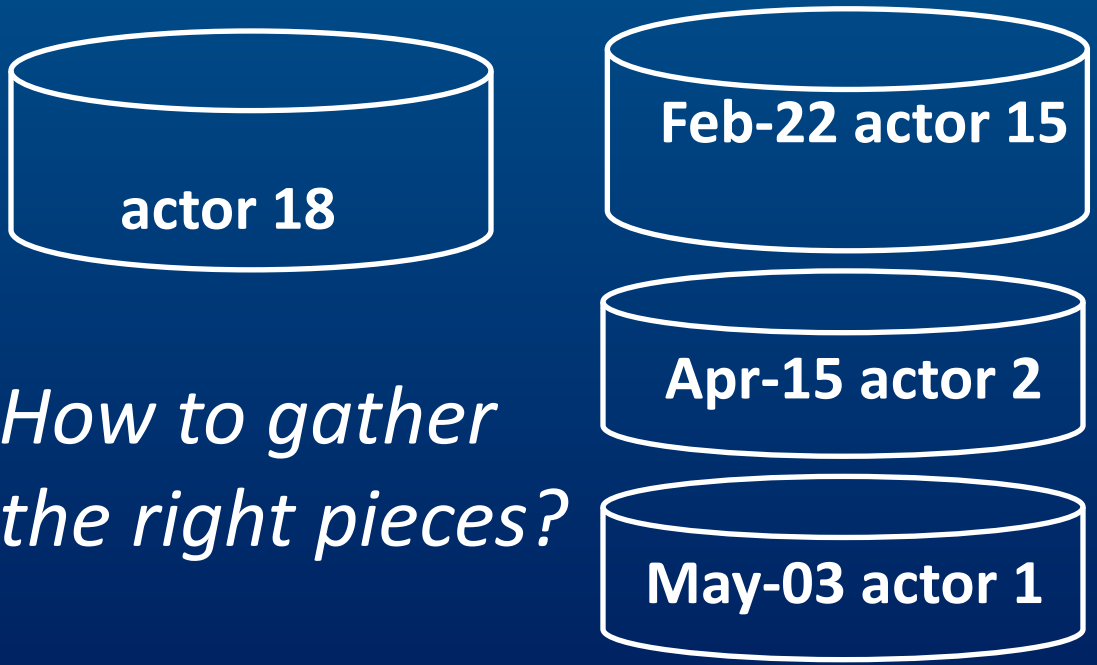
- Result wanted:

File AjoinB: <word date, day-count total-count >

```
able    Jan-16,    2   5
actor   Feb-22,    15   18
actor   May-03,    1    18
burger  Jul-04,    20   25
.
.
.
```

Joining Data

- Recall that data is split in parts



How to gather the right pieces?

Key-Value & Task Decomposition

- Main design consideration:

Join depends on word

*(e.g. Select * where A.word=B.word)*

Key-Value & Task Decomposition

- For the join:
  - Let <key> = word
  - Let <value> = other info

<*word,* >

Key-Value & Task Decomposition

- Note:

File A: <word, total-count>

```
able ,   5
actor ,  18
. . .
```

File B: <date word, day-count>

```
Jan-16  able ,   2
Feb-22  actor ,  15
. . .
```

Key-Value & Task Decomposition

- Note:

File A: <word, total-count>

```
able ,   5
actor ,  18
. . .
```

File B: <date word, day-count>

```
Jan-16  able ,   2
Feb-22  actor ,  15
. . .
```

word already the key

Key-Value & Task Decomposition

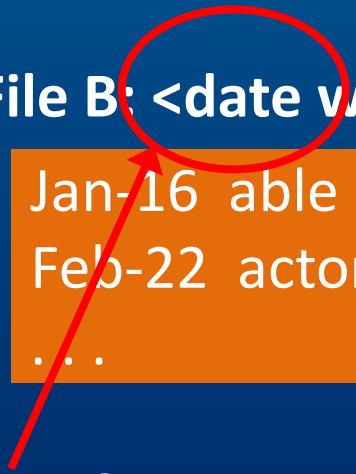
- Note:

File A: <word, total-count>

```
able ,   5
actor ,  18
. . .
```

File B: <date word, day-count>

```
Jan-16  able ,   2
Feb-22  actor ,  15
. . .
```



date needs to be filtered out

Key-Value & Task Decomposition

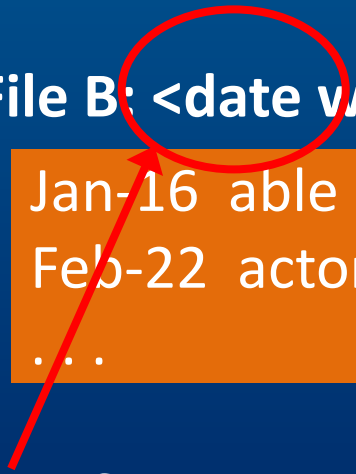
- Note:

File A: <word, total-count>

```
able ,   5
actor ,  18
. . .
```

File B: <date word, day-count>

```
Jan-16  able ,   2
Feb-22  actor ,  15
. . .
```



date needs to be filtered out

Where should date info go?

Key-Value & Task Decomposition

<word, date day-count total-count >

put date into value field

Task Decomposition

- Now data sets are:

File A: <word, total-count>

able , 5
actor , 18
burger , 25
.
.
.

File B_new: <word, date count>

able , Jan 16 2
actor , Feb-22 15
actor , May-03 3
burger , Jul-04 20
.
.
.

Task Decomposition

- How will Hadoop shuffle & group these?

File A: <word, total-count>

able , 5
actor , 18
burger , 25
.
.
.

File B_new: <word, date day-count>

able , Jan-16 2
actor , Feb-22 15
actor , May-03 3
burger , Jul-04 20
.
.
.

Task Decomposition

- How will Hadoop shuffle & group these?

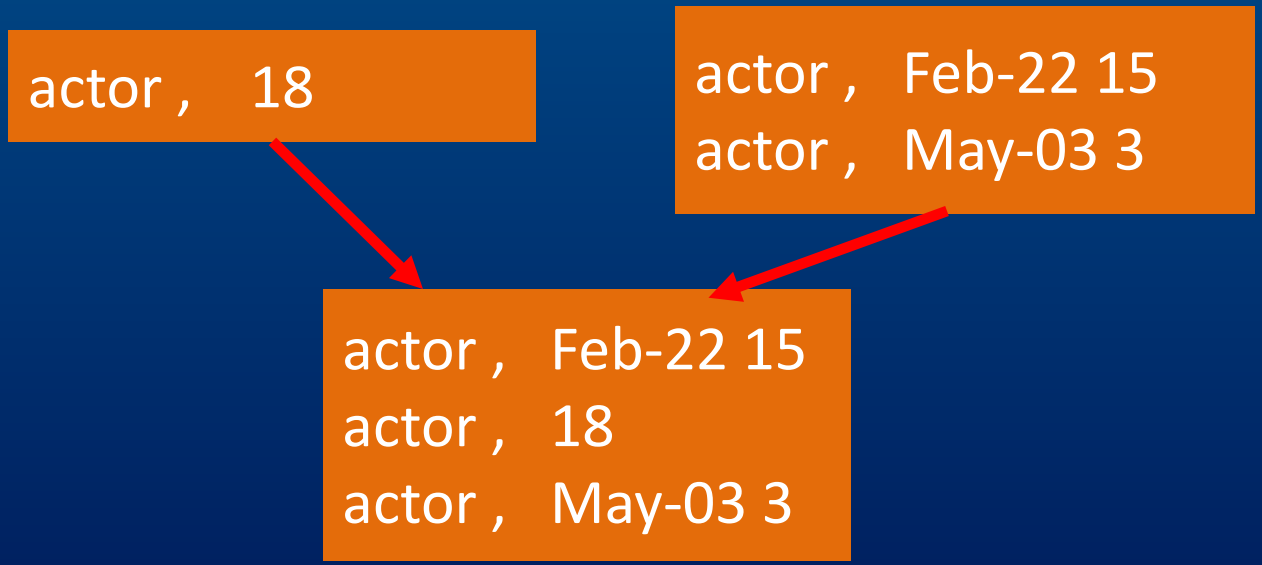
Let's focus on 1 key:

actor , 18

actor , Feb-22 15
actor , May-03 3

Task Decomposition

- Hadoop gathers the data for a join



Task Decomposition

- Reducer now has all the data for same word grouped together

Task Decomposition

- Reducer can now join the data and put date back into key

Example:
Vector Multiplication

Vector Multiplication

- Task: multiply 2 arrays of *N* numbers
  - A basic mathematical operation
  - Let's assume *N* is very large

Vector Multiplication

- Task: multiply 2 arrays of *N* numbers

A	X	***B***
5		2.7
4		1.9
-3.2		-1.3
.		.
.		.
.		.
-2		1

$=$

(5×2.7) # 1st of *A* & *B*

$+ (4 \times 1.9)$ # 2nd of *A* & *B*

$+ (-3.2 \times -1.3)$ # 3rd …

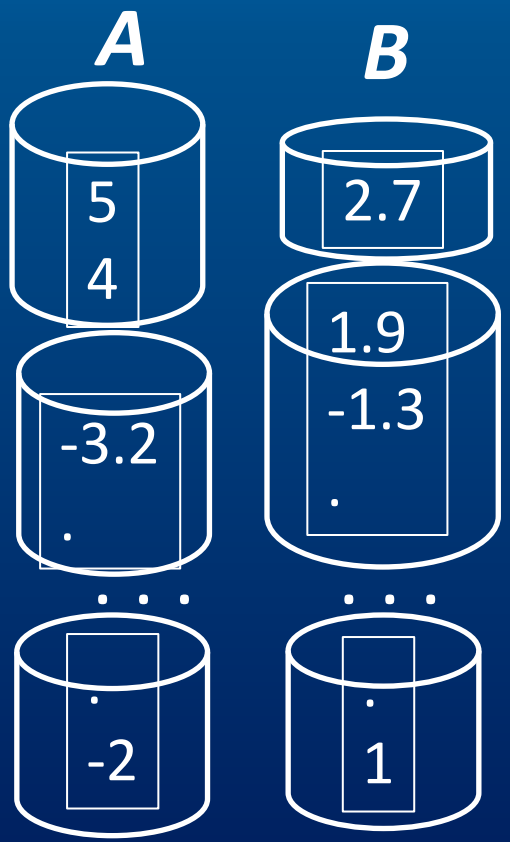
.

.

.

$+ (-2 \times 1)$ # Nth of *A* & *B*

Vector Multiplication



- Recall:
  - data partitioned in HDFS

Vector Multiplication

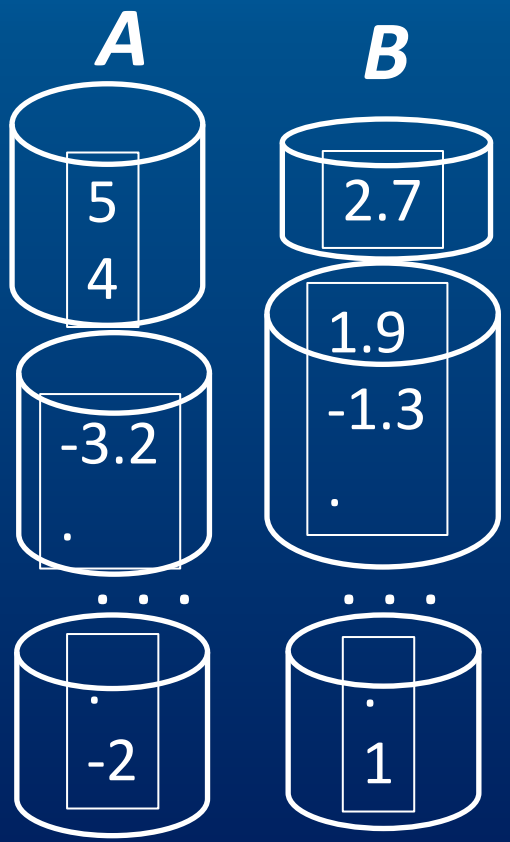
- Main design consideration:

need elements with same index together

Let <key, value> =

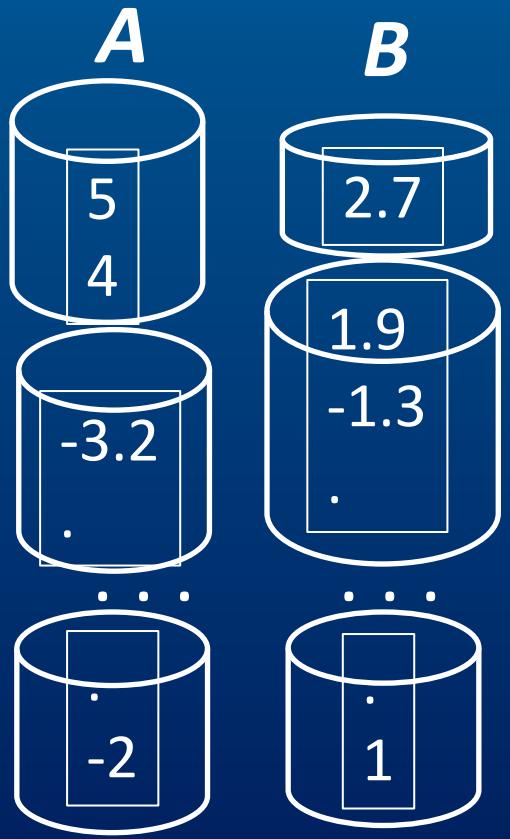
<index, number>

Vector Multiplication



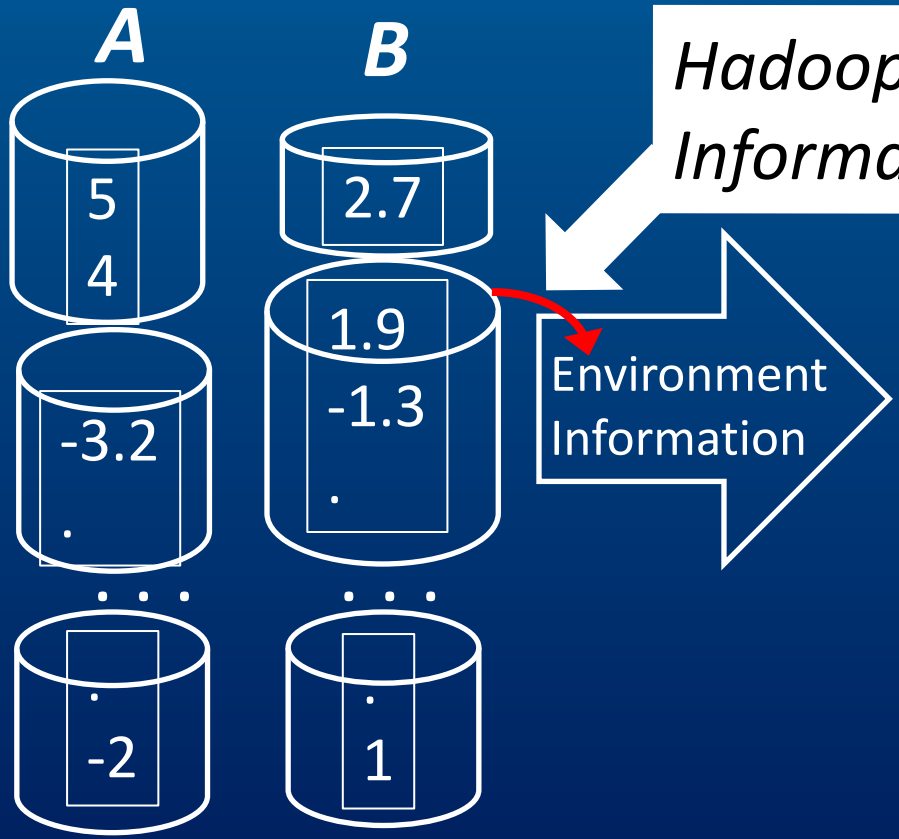
- Problem: array partitions don’t have an index

Vector Multiplication



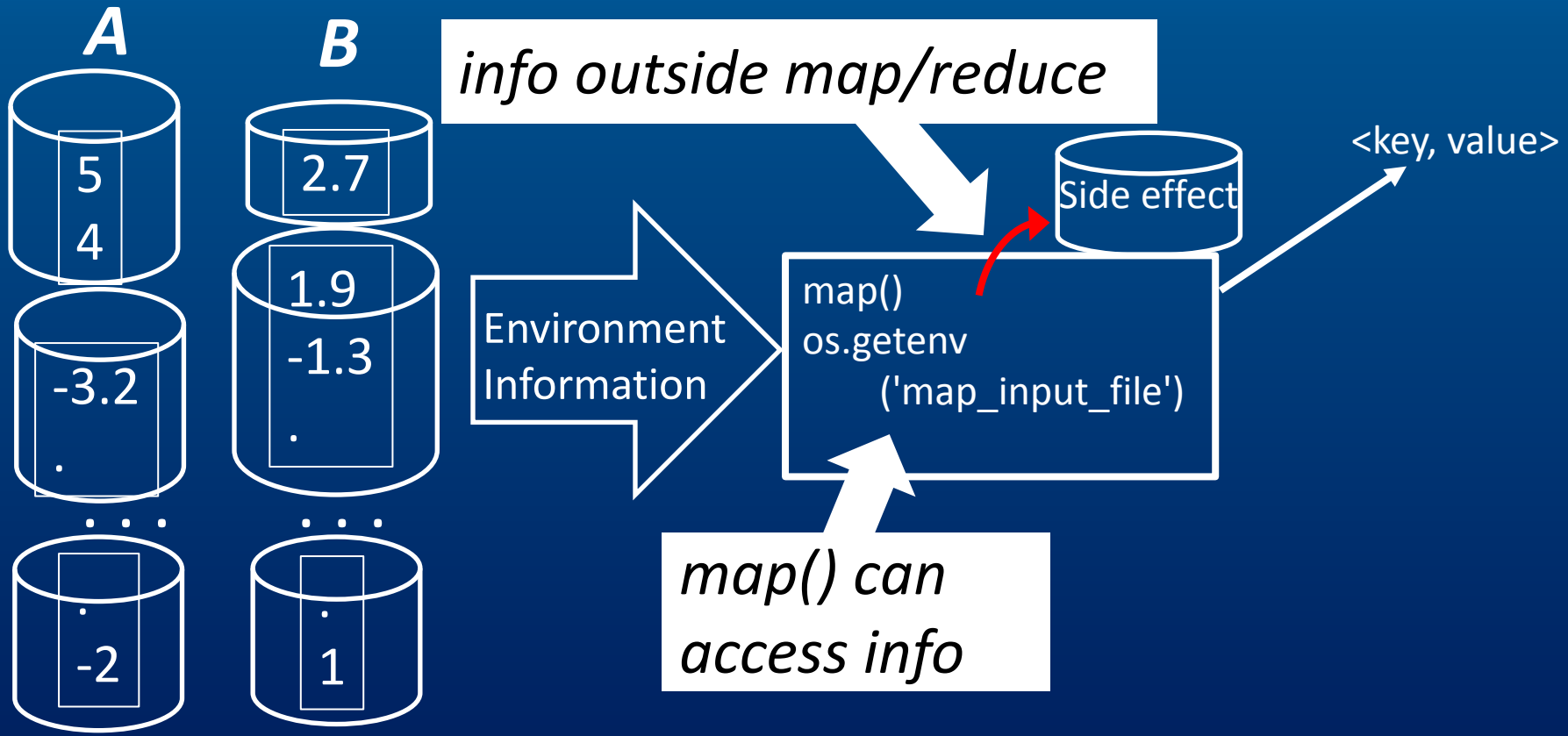
- Problem: array partitions don't have an index

Vector Multiplication

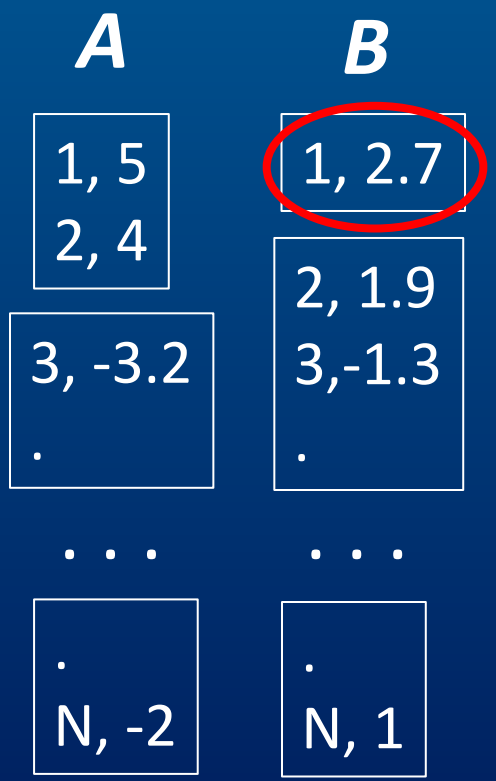


Hadoop makes environmental Information available

Vector Multiplication

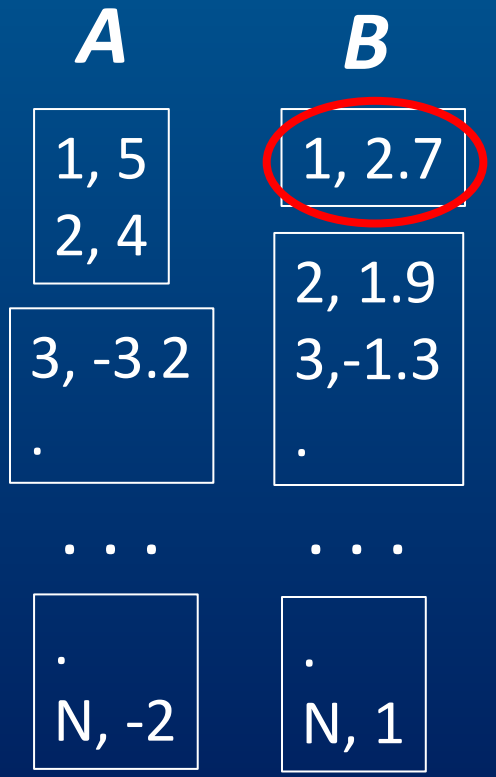


Vector Multiplication



- Let's assume:
  - each line already has <index, number>

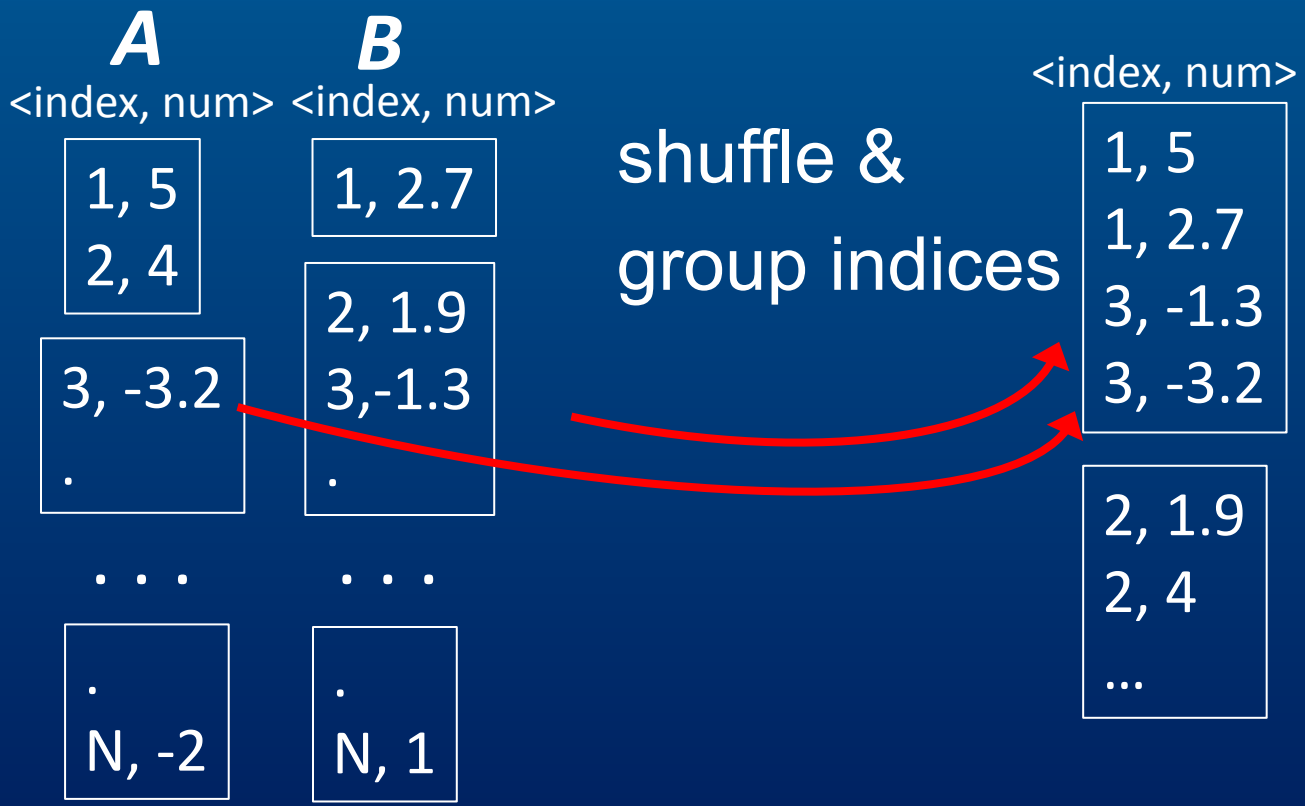
Vector Multiplication



- Let's assume:
  - each line already has <index, number>

Note: mapper only needs to pass data (identity function)

Vector Multiplication



Vector Multiplication

A,B grouped

<index, num>

1, 5
1, 2.7
3, -1.3
3, -3.2

2, 1.9
2, 4

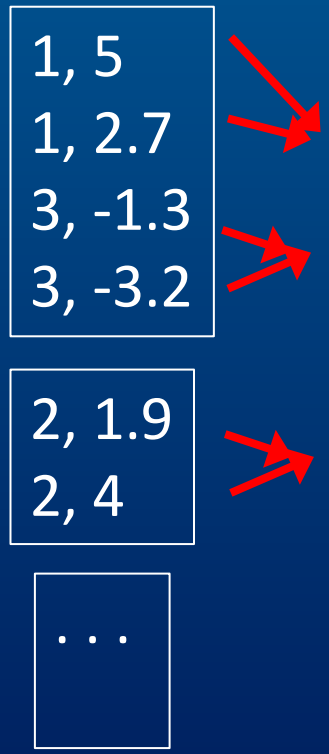
. . .

What should reducers do?

Vector Multiplication

A,B grouped

<index, num>



Reducer:

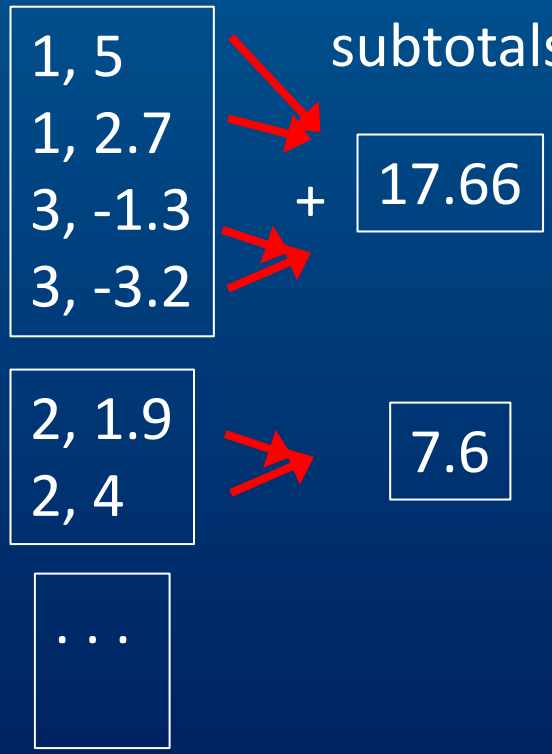
-get pairs of

<index, number>

Vector Multiplication

A,B grouped

<index, num>



Reducer:

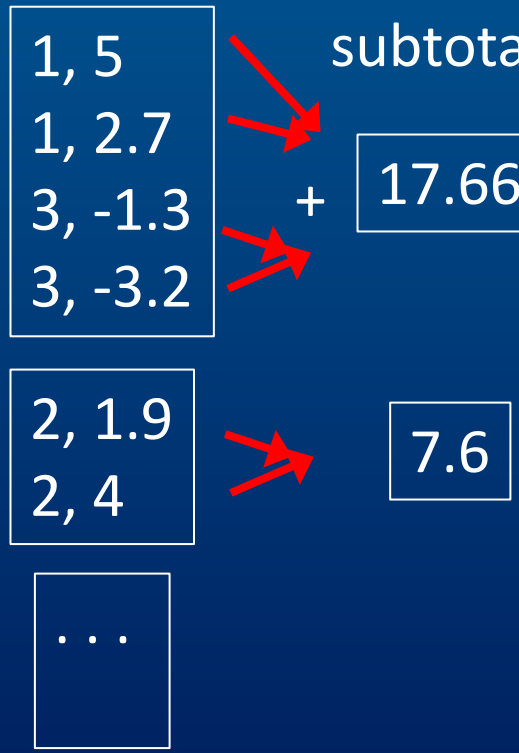
-get pairs of <index, number>

-multiply & add

Vector Multiplication

A,B grouped

<index, num>



Reducer:

-get pairs of <index, number>

-multiply & add

(Still need get total sum, but should be largely reduced)

Computational Costs

In Vector Multiplication

Computational Costs

- For Vector Multiplication
  - *How many <index, number> are output from map()?*

Computational Costs

- For Vector Multiplication
  - *How many <index, number> are output from map()?*
  - *How many <index> groups have to be shuffled?*

Computational Costs

- How many <index, number> are output ?

A	***B***
1, 5	1, 2.7
2, 4	2, 1.9
3, -3.2	3, -1.3
.	.
.	.
.	.
N, -2	N, 1

For: 2 Vectors with
N indices each
Then:
2*N* <index, number>
are output from map()

Computational Costs

- How many <index> groups have to be shuffled?

A,B grouped

<index, num>

1, 5
1, 2.7
3, -1.3
3, -3.2

2, 1.9
2, 4
…

For: $2N$ indices and
N pairs

Then:

N groups are shuffled to reducers

Computational Costs

- Can we reduce shuffling?

Computational Costs

- Can we reduce shuffling?

- Try: ‘combine’ map indices in mapper
  (works better for Wordcount)

Computational Costs

- Can we reduce shuffling?
- Or Try: use index ranges of length R

Computational Costs

- Index Ranges: let R=10 & bin the array indices

1 2 3 4 … 10 11 12 …..19 20 21 …. (N-9) …. N

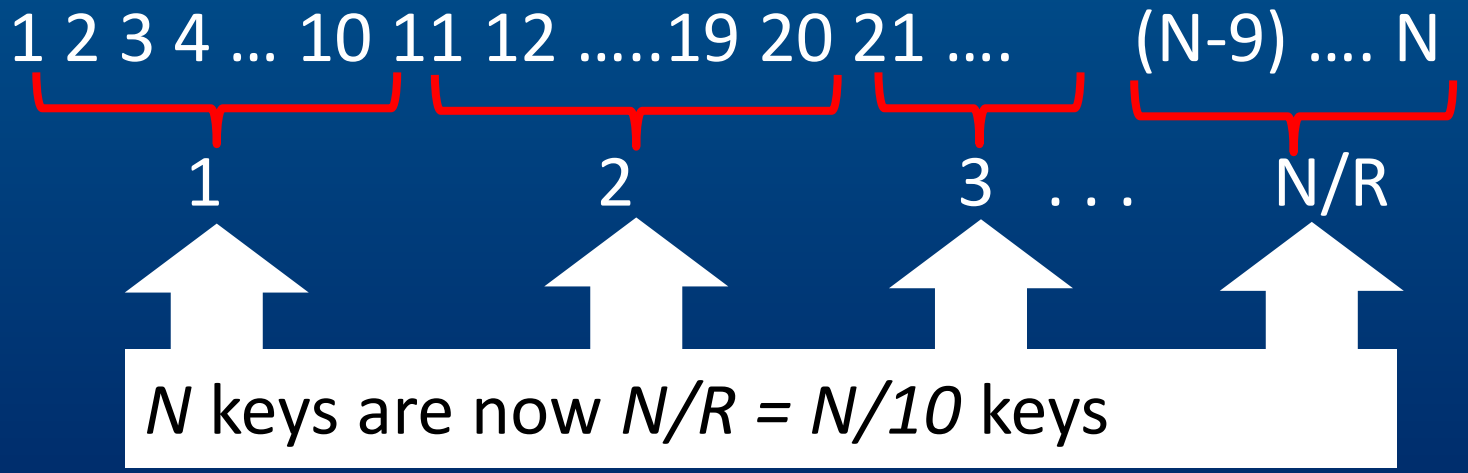
N keys

Computational Costs

- Index Ranges: let R=10 & bin the array indices

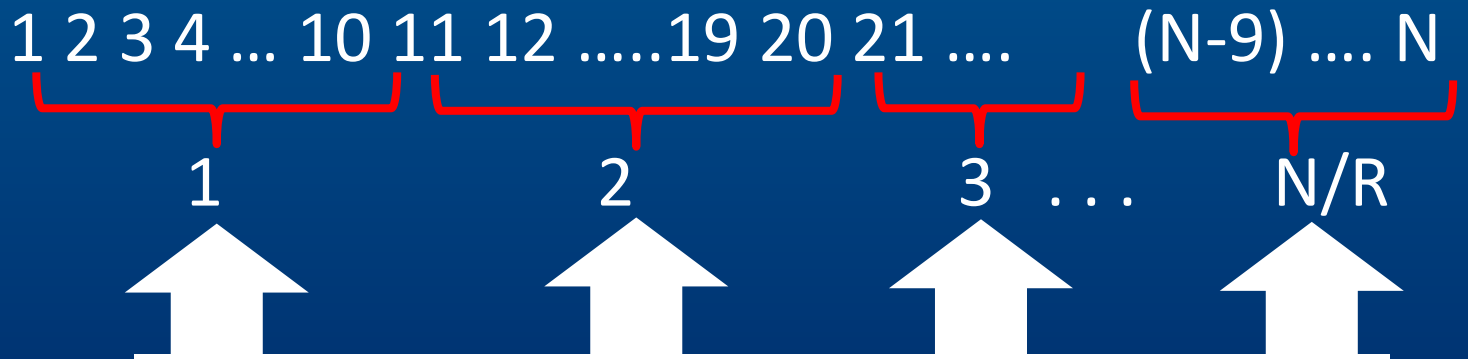
Computational Costs

- For example, let R=10, and bin the array indices



Computational Costs

- For example, let R=10, and bin the array indices



N keys are now *N/R=N/10* keys

<key,value> is now

<index bin, original-index number>

Computational Costs

- Now shuffling costs depend on N/R groups

If: *R=1*
Then: *N/R=N groups (same as before)*

If: *R>1*
Then: *N/R<N (less shuffling to do)*

Computational Costs

- Trade-offs:

If:
    size of (N/R) ↑
Then:
    shuffle costs ↑

Computational Costs

- Trade-offs:

If:
    size of (N/R) ↑
Then:
    shuffle costs ↑
But:
    reducer complexity ↓

Computational Costs

- Trade-offs:

If:
size of (N/R) ↑
Then:
shuffle costs ↑
But:
reducer complexity ↓

-you control R
(specific tradeoffs
depend on data
and hardware)

Vector to Matrices

- Matrix multiplication needs row-index and col-index in the keys
- Matrix multiplication more pertinent to data analytic topics

Summary

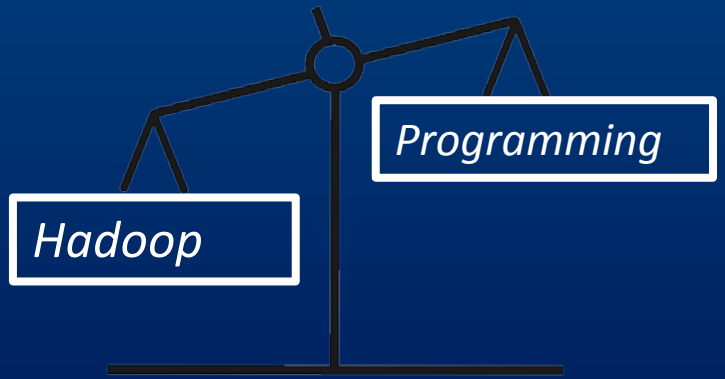
And Looking Beyond

Task Decomposition

- mappers are separate and independent
- mappers work on data parts

Summary of Map/Reduce Design Considerations

- <key, value> must enable correct output
- Let Hadoop do the hard work
- Trade-offs



Summary of Map/Reduce Design Considerations

- Common mappers:
  - Filter (subset data)
  - Identity (just pass data)
  - Splitter (as for counting)

Summary of Map/Reduce Design Considerations

- Composite <keys>

Summary of Map/Reduce Design Considerations

- Composite <keys>
- Extra info in <values>

Summary of Map/Reduce Design Considerations

- Composite <keys>
- Extra info in <values>
- Cascade Map/Reduce jobs

Summary of Map/Reduce Design Considerations

- Composite <keys>
- Extra info in <values>
- Cascade Map/Reduce jobs
- Bin keys into ranges

Summary of Map/Reduce Design Considerations

- Composite <keys>
- Extra info in <values>
- Cascade Map/Reduce jobs
- Bin keys into ranges
- Aggregate map output when possible (combiner option)

Potential Limitations Map/Reduce

- Must fit <key, value> paradigm
- Map/Reduce data not persistent
- Requires programming/debugging
- Not interactive

Beyond Map/Reduce

- Data access tools (Pig, HIVE)
  - SQL like syntax
- Interactivity & Persistency (Spark)