Lesson 4 Notes

Designing With Patterns

My name is Andy, I'm going to be presenting this lesson, where we are going to learn about "design patterns" in mapreduce. The patterns that we will cover are templates for solving common problems with mapreduce. These patterns achieve a good balance between flexibility and rigidity: They are general enough that they can be adapted to solve lots of problems, but specific enough that they don't require too much effort to use. The goal here is to give you familiarity (but not necessarily mastery) with some of these patterns. We won't discuss where these patterns came from or why they were chosen. We are just going to introduce them. Note: these patterns come from the book: Map Reduce Design Patterns by Donald Miner & Adam Shook.

What we will Cover

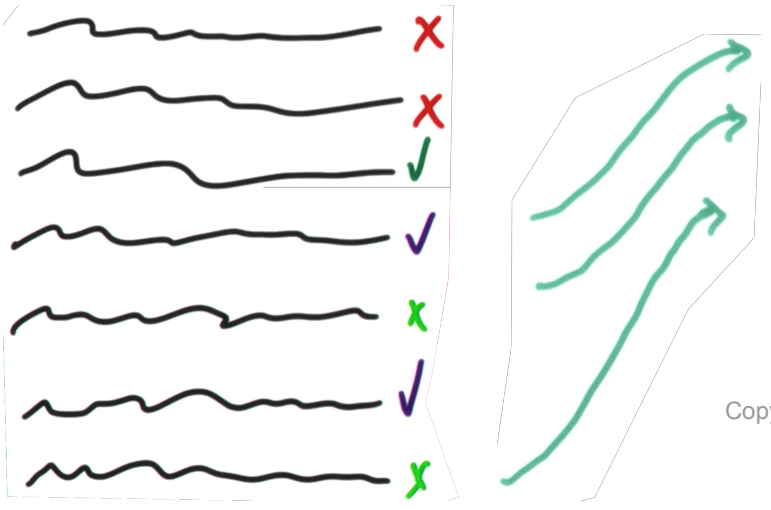
Not every problem can be solved with MapReduce. In fact, problems have to be manipulated to fit into this framework of mapping and reducing. But luckily programmers have developed a sophisticated arsenal of patterns to make some potentially-unexpected problems "mapreducable." In this lesson, we'll introduce the following "patterns."

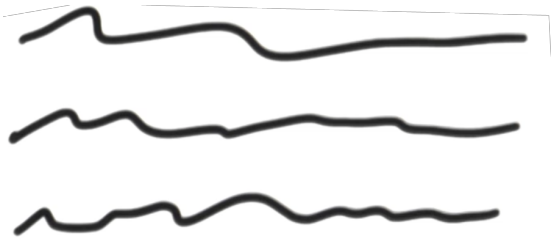
- Filtering Patterns: don't change records. Only get a part of the data!
    - Examples: sampling, random-sampling, top 10
- Summarization Patterns: Give top level view of data.
    - Examples: counting, minimum, maximum, mean, median, standard deviation, inverted index
    - Take a break to discuss combiners.
- Structured-to-Hierarchical Patterns:
    - Examples: combining 2 datasets.

This lesson will go fast. You probably won't remember everything about these patterns, but that's okay. You'll know that they exist and you'll be able to consult them when you find yourself facing a problem that you think could be molded to fit one of these patterns.

Filtering Patterns

These patterns have one thing in common: they don't change the actual records. Records that pass the filter, are output exactly the way they came in. These patterns all find a subset of data, whether it be small, like a top-ten listing, or large, like all results from the last year from a dataset that contains records for 5 years. There are several possible applications of filtering patterns:



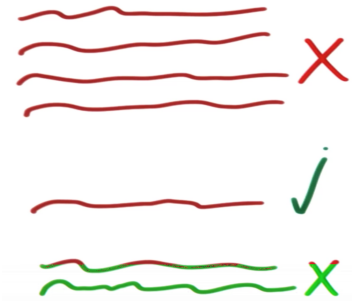


A simple filtering boils down to having a function that given an input returns “keep” or “throw away”. Mapper applies this function to every input element, and accordingly the output will be a filtered subset of the input containing only data that is interesting to you

Sampling is about pulling out a subset of the data for future processing. Typically, you'd use sampling to extract a smaller but representative data set on which you could then perform further analysis. However, taking just the first 10,000 records from a massive dataset might not be the best idea because it might not be representative. Instead, you could use Python's random number generator to return just, say, 1% of the data by only passing it through the Mapper if a random value between 1 and 100 equalled 1.

Top 10: very interesting case, we all love to see top lists!

Sampling



Lets have some practice with filtering! This time we will work with a dataset that is generated by our very own Udacity students - our forums! We are interested in getting a subset of data from the forum that contains only posts that are 1 sentence or less (so, contains only one period/!/? character, and period is the last significant character in the post).

Top 10

An interesting application of mapreduce is making top N record lists. In RDBMS you would normally first sort the data, then take top N records. In mapreduce this kind of approach will not work, because the data is not sorted and is processed on several machines. Thus, the mappers will first have to find their own top N lists, without sorting the data, and then send the local lists to the reducers who then can find the global top N list.

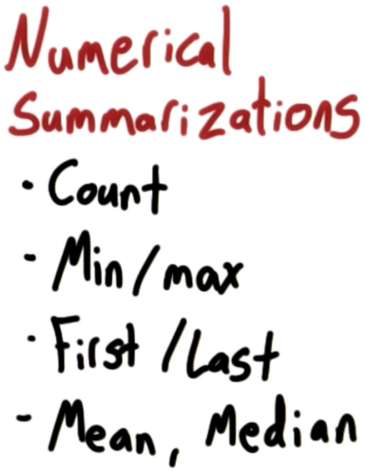
Let's find the top 5 longest posts in our forum!

Summarization Patterns

The next set of design patterns we will discuss will be patterns that produce a top-level, summarized view of your data. This is very powerful approach to get a birds eye view of your data, something that you can not get by just manually looking at parts of the data. You can group similar data together, for example by day, or by time of day, or by username and then calculate a statistic of some value, like min/max/average/mean etc. You can also build an index, or just simply count occurrences of something.

There are couple of problems that can be solved by using similar approach:

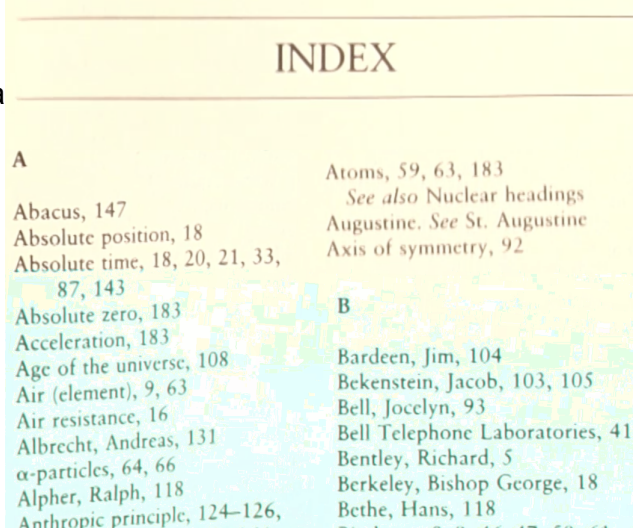
- numerical summarizations - finding count of value, as well as min, max, avg, median and other numerical values for your dataset
- inverted index: important when building a search engine or full text search functionality for your own website or application.



You can use built in Hadoop functionality to perform some

of these operations more efficiently and we talk about that as well.

Inverted Index



INDEX

A

Abacus, 147
Absolute position, 18
Absolute time, 18, 20, 21, 33, 87, 143
Absolute zero, 183
Acceleration, 183
Age of the universe, 108
Air (element), 9, 63
Air resistance, 16
Albrecht, Andreas, 131
α-particles, 64, 66
Alpher, Ralph, 118
Anthropic principle, 124–126,
Atoms, 59, 63, 183
See also Nuclear headings
Augustine. *See* St. Augustine
Axis of symmetry, 92

B

Bardeen, Jim, 104
Bekenstein, Jacob, 103, 105
Bell, Jocelyn, 93
Bell Telephone Laboratories, 41
Bentley, Richard, 5
Berkeley, Bishop George, 18
Bethe, Hans, 118

It is often needed to build a reverse index from a data set, to enable faster searching. The obvious example would be a web search engine. You need to create a mapping from keywords to web links, to enable faster finding of relevant information. Think of it as an index for a book - you have a word, or term, and all pages you can find this term.

While the pattern itself is simple - mapper outputs each word as the key, the link being the value, you have to be conscious of the fact that this type of mapreduce is susceptible to uneven key distribution. For example, you can assume that the word “the” will be found a lot more in a text than other words. You have to think if you really need to include such words in the index.

Numerical summarizations

Common uses for this type of analysis are:

- word or record count (that you already used when analyzing webserver logfiles). Mapper just outputs the thing that you are interested as a key, and 1 as a value. Reducer then can just sum up the values. Another very popular example is word count, which might be the canonical “Hello, World” for MapReduce - count how many times a word appears in a dataset.

Word/record Count

Key: thing-u-want
Value: 1

- min/max/count for a particular event, for example first/last time a user posted on a forum, or first/last time and how many particular item was bought in a shop.

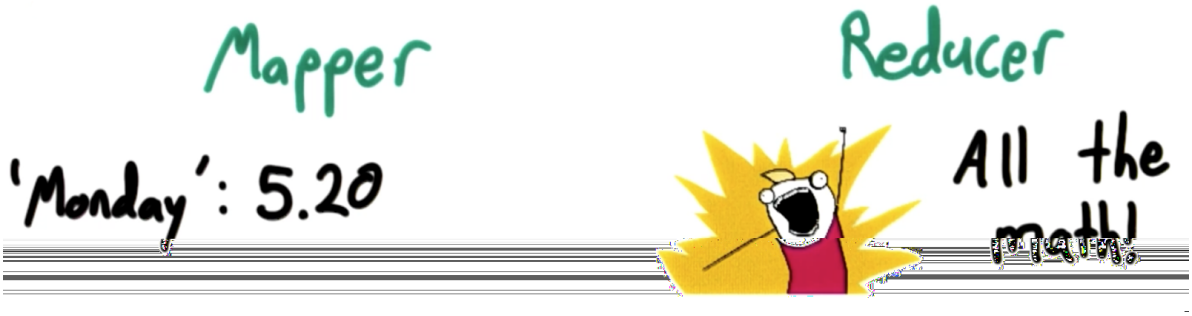
Mean Median Std. Dev.

Mean and Standard Deviation

Let's say that you want to know if there is any correlation between day of week and how much money people spend on items. Your task is to find out mean and standard deviation for sale per day of week.

Mapper will need to have only output the day of week as the key and the sale amount as the value. Reducer has to do all the maths.

In general - if you need to find out Average/Median/Standard deviation, these would have to be calculated in the reducer.



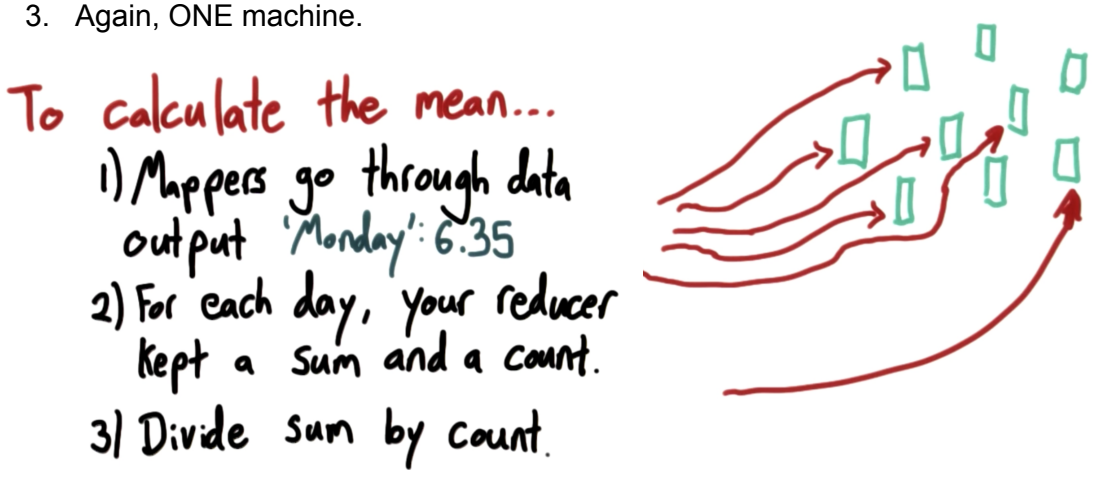
Combiners

To calculate the mean in that last problem, what did you do? You may have done something like this:

1. Your mapper may have gone through the records and output a key; value pair that looked like: day of week; value.
2. For each day of the week, your reducer kept a running total of the value as well as a count of the number of records.
3. You divided the total value by the number of records to get the mean.

Now, let's think about WHERE each of these steps took place.

1. On various machines throughout the network.
2. On ONE machine in the network.
3. Again, ONE machine.



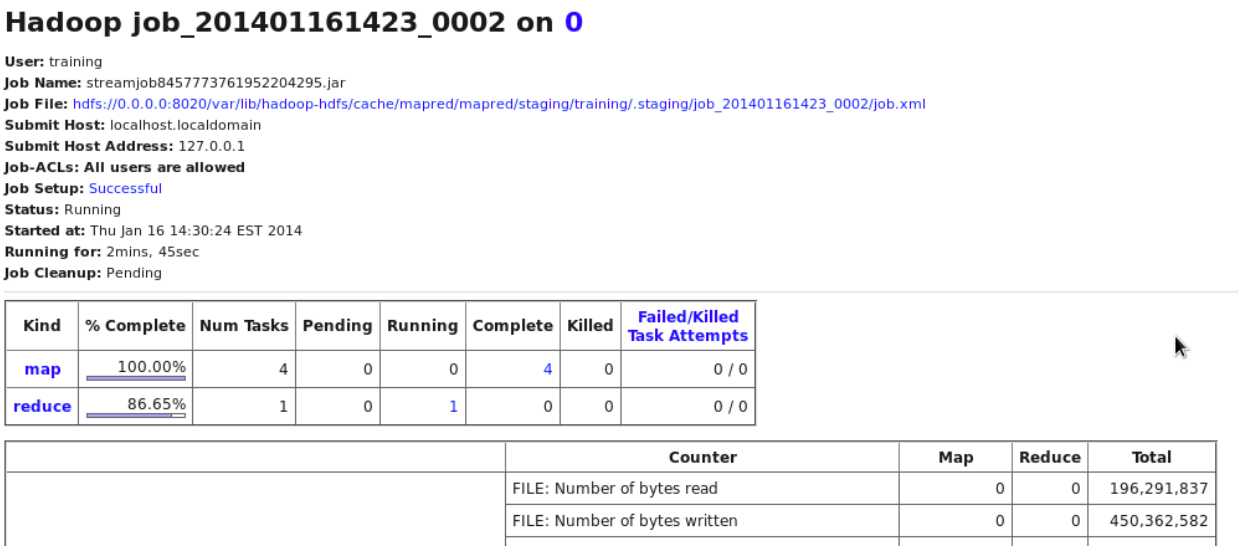
But there's a problem here. That second step involves moving a lot of data around your network. What if we could do some of the reduction locally before sending the data to the reducers?

We can! We can use combiners! Combiners will, in essence, do as much reduction as possible locally before sending that data to the reducers.

- This might save significant network traffic in you have a lot of records, but much less unique keys. You will need to add additional command to the command line script to use this functionality or use the full java command. Please see instructor comments for detailed information.
- When you run a job, you will see some output on the screen, which includes a tracking url:

```
14/01/16 14:30:24 INFO streaming.StreamJob: UNDEF/bin/hadoop job  -Dmapred.job.tracker=0.0.0.0:8021 -kill job_20
1401161423_0002
14/01/16 14:30:24 INFO streaming.StreamJob: Tracking URL: http://0.0.0.0:50030/jobdetails.jsp?jobid=job_20140116
1423_0002
14/01/16 14:30:25 INFO streaming.StreamJob:  map 0%  reduce 0%
14/01/16 14:30:36 INFO streaming.StreamJob:  map 12%  reduce 0%
```

- Open it in a browser
- You will see a job page, containing information about the job, as it is being run



Hadoop job_201401161423_0002 on 0

User: training
Job Name: streamjob8457773761952204295.jar
Job File: hdfs://0.0.0.0:8020/var/lib/hadoop-hdfs/cache/mapred/mapred/staging/training/.staging/job_201401161423_0002/job.xml
Submit Host: localhost.localdomain
Submit Host Address: 127.0.0.1
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Running
Started at: Thu Jan 16 14:30:24 EST 2014
Running for: 2mins, 45sec
Job Cleanup: Pending

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	4	0	0	4	0	0 / 0
reduce	86.65%	1	0	1	0	0	0 / 0

	Counter	Map	Reduce	Total
	FILE: Number of bytes read	0	0	196,291,837
	FILE: Number of bytes written	0	0	450,362,582

- Here are the comparison screenshots from 2 jobs, one run without combiner and one with a combiner:

Map input records	0	0	4,138,476
Map output records	0	0	4,138,476
Map output bytes	0	0	67,381,890
Input split bytes	0	0	420
Combine input records	0	0	0
Combine output records	0	0	0
Reduce input groups	0	0	103
Reduce shuffle bytes	0	0	75,658,866
Reduce input records	0	0	4,138,476
Reduce output records	0	0	103
Spilled Records	0	0	12,219,835
CPU time spent (ms)	0	0	43,420
Physical memory (bytes) snapshot	0	0	903,065,600
Virtual memory (bytes) snapshot	0	0	1,941,934,080
Total committed heap usage (bytes)	0	0	772,935,680
BYTES_READ	211,312,924	0	211,312,924

Map input records	0	0	4,138,476
Map output records	0	0	4,138,476
Map output bytes	0	0	67,381,890
Input split bytes	0	0	420
Combine input records	0	0	4,140,330
Combine output records	0	0	2,266
Reduce input groups	0	0	103
Reduce shuffle bytes	0	0	9,609
Reduce input records	0	0	412
Reduce output records	0	0	103
Spilled Records	0	0	2,678
CPU time spent (ms)	0	0	31,420
Physical memory (bytes) snapshot	0	0	798,273,536
Virtual memory (bytes) snapshot	0	0	1,944,887,296
Total committed heap usage (bytes)	0	0	657,997,824
BYTES_READ	211,312,924	0	211,312,924

As you can see, when using a reducer (second screenshot), reducers get significantly less records and have to shuffle less bytes than without a combiner. Without combiner - 4,138,476 records, with combiner - 412 records. While it does not lead to time savings on a single node pseudodistributed cluster run in a VM, like you have, in real world it could save a lot of network traffic.

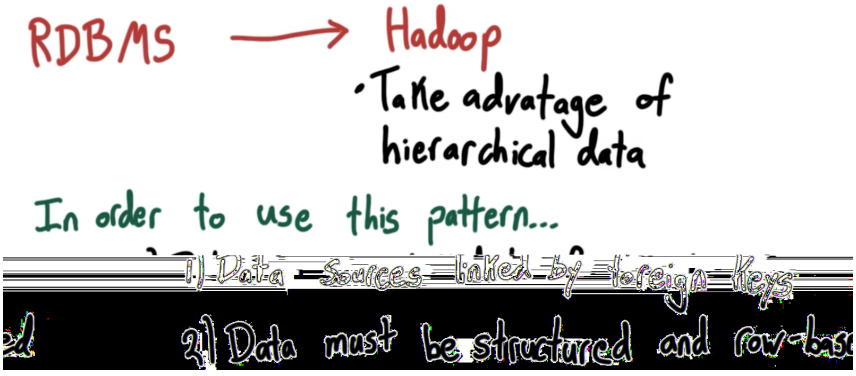
Generally, we want our combiners to do the same thing as our reducers. Also, keep in mind that this intermediate step may introduce some complications when we try to do our computation. You'll encounter that in the next exercise.

Calculating the Sum with Combiners

Try calculating the sum of values, like you did in lesson 3, but this time use a combiner.

Structured to Hierarchical Patterns

When migrating data from an RDBMS to a Hadoop system, one of the first things you should consider doing is reformatting your data. Since Hadoop doesn't care what format your data is in, you can take advantage of hierarchical data to avoid doing live joins of several datasets at the time of analysis. If you know what kind of information you will



want to get later, you might save significant time by reformatting the data.

You can use this pattern if you have data sources that are linked by some set of foreign keys and your data is structured and row-based.

Combine 2 datasets

You might want to know what the reputation of the author of a post is. Combine post and user tables to produce a simple dataset that has all the relevant information together, or in RDBMS terms - is denormalized.

Post table will contain information like this:

Post:

id	title	tagnames	author_id	body
6120	Homework assignments	cs101 homework	100001360	I notice that if I go back and review the answ...
3641	Homework 1[Spanish Group]	cs101 spanish homework hw1	100000800	Dudas y consultas sobre la tarea numero 1.

User table will have this:

User:

user_ptr_id	reputation	gold	silver	bronze
100022094	6354	4	12	50
100071633	1468	5	9	38
100027859	2508	4	18	59
100020261	219	1	1	4

We want to combine the tables, so that each post contains information about the reputation and badges for the author, so that we can run some analysis on content of a post in relation to reputation of a user. For example, is there a correlation between post length and user reputation?

Conclusion

We've covered several patterns … what good are they?

Well, you'll have to practice with them to have them become second nature, but for now you can ask yourself when solving a problem: "Does this problem fit into one of the patterns I learned? Is it a filtering problem? A summarization problem? a Hierarchical data problem?"

These are not all the patterns you may want to use. And getting yourself to ask this question is

not simple. In fact, just identifying when mapreduce is the appropriate tool for a problem is a large part of what it means to be a mapreduce expert.

If you want to gain this expertise, you'll have to practice. This may happen naturally in a job environment, or you could make it happen deliberately by seeking out problems and trying to solve them with this framework.

If you enjoy learning about these patterns, you can find more in the book …

Nice work and congratulations. Now, on to the final project! Good luck.