Map Reduce

Data at Scale

History

- A *simple* paradigm that popped up several times as paradigm
- Observed by google as a software pattern:
    - Data gets filtered locally and filtered data is then reassembled elsewhere
        - Software pattern: Many engineers are re-engineering the same steps
- Map-reduce:
    - Engineer the common steps efficiently
    - Individual problems only need to be engineered for what makes them different

History

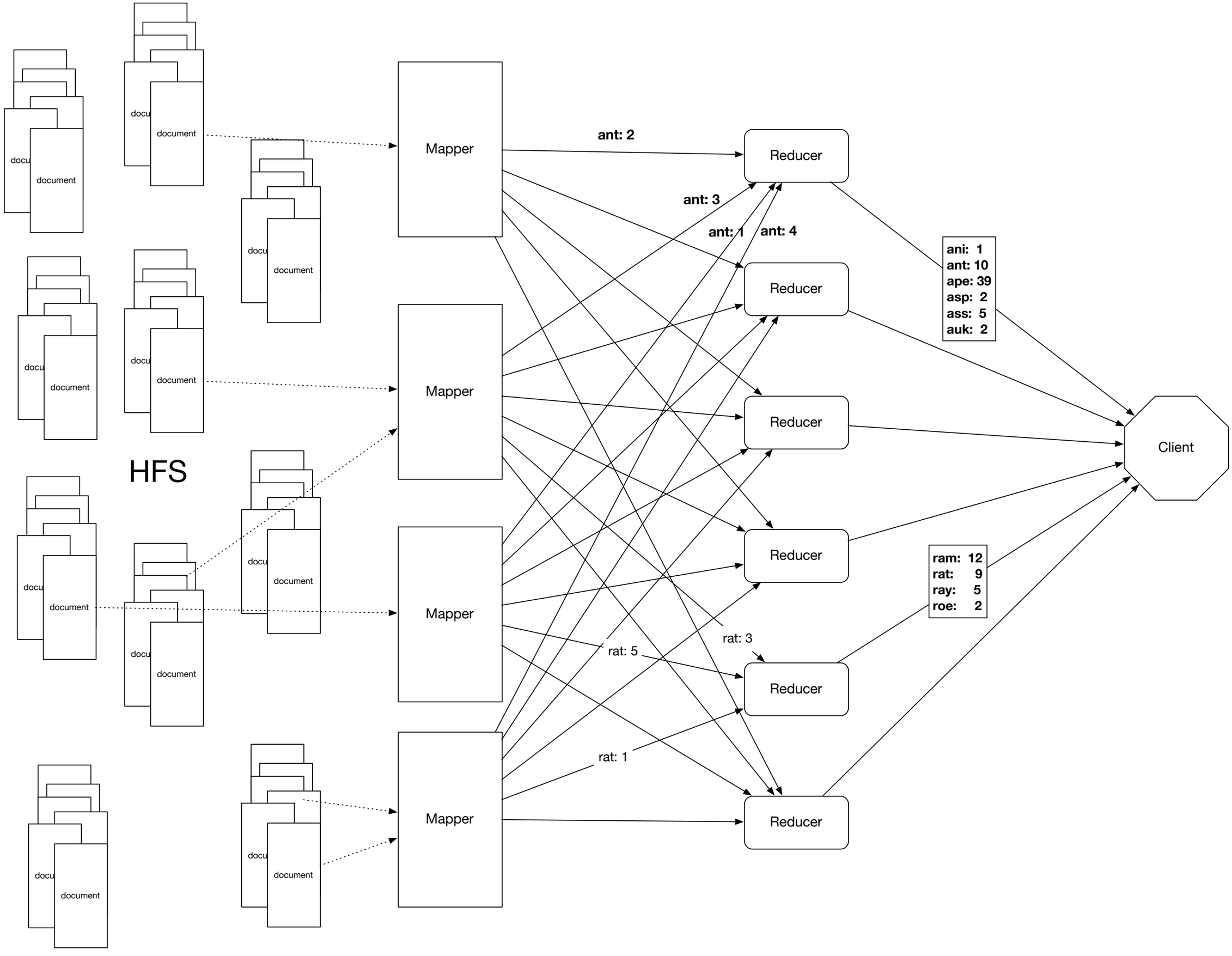
- Open source project (in part sponsored by Yahoo!)
  - Java-based Hadoop
  - Eventually a first tier Apache Foundation project
- Other projects at higher level: Pig, Hive, HBase, Mahout, Zookeeper
  - Use Hadoop as foundation
  - Hadoop is becoming a distributed OS

Map Reduce Paradigm

- Input: Large amount of data spread over many different nodes
- Output: A single file of results
- Two important phases:
  - **Mapper**: Records are processed into key-value pairs. Mapper sends key-value pairs to reducers
  - **Reducer:** Create final answer from mapper

Simple Example

- Hadoop Word Count
  - Given different documents on many different sites
    - Mapper:
      - Extract words from record
      - Combines words and generates key-value pairs of type word: key
      - Sends to the reducers based on hash of key
    - Reducer:
      - Receives key-value pairs
      - Adds values for each key
      - Sends accumulated results to aggregator - client


HFS
docu
document
Mapper
Mapper
Mapper
Mapper
ant: 2
ant: 3
ant: 1
ant: 4
rat: 5
rat: 3
rat: 1
Reducer
Reducer
Reducer
Reducer
Reducer
ani: 1
ant: 10
ape: 39
asp: 2
ass: 5
auk: 2
ram: 12
rat: 9
ray: 5
roe: 2
Client

Map-reduce paradigm in detail

- The simple mapper -reducer paradigm can be expanded into several, typical components

Map Reduce in Detail

- **Mapper:**
  - Record Reader
    - Parses the data into records
    - Example: Stackoverflow comments.
      - `<row Id="5" PostId="5" Score="2" Text="Programming in Portland, cooking in Chippewa ; it makes sense that these would be unlocalized. But does bicycling.se need to follow only that path? I agree that route a to b in city x is not a good use of this site; but general resources would be." CreationDate="2010-08-25T21:21:03.233" UserId="21" />`
    - Record reader extract the “Text=” string
    - Passes record into a key-value format to rest of mapper

Map Reduce in Detail

- **Mapper**
  - map
    - Produces “intermediate” key-value pairs from the record
  - Example:
    - **"Programming in Portland, cooking in Chippewa ; it makes sense that these would be unlocalized. But does bicycling.se need to follow only that path? I agree that route a to b in city x is not a good use of this site; but general resources would be.”**
    - Map produces: <programming: 1> <in: 1> <Portland: 1> <cooking: 1> <in: 1> …

Map Reduce in Detail

- **Mapper**
  - Combiner — a local reducer
    - Takes key-value pairs and processes them
    - Example:
      - Map produces: <programming: 1> <in: 1> <Portland: 1> <cooking: 1> <in: 1> …
      - Combiner combines words: <programming: 1> <in: 4> <Portland: 3> …

Map Reduce in Detail

- Combiners allow us to reduce network traffic
  - By compacting the same infomrmation

Map Reduce in Detail

- **Mapper**
  - Partitioner
    - Partitioner creates shards of the key-value pairs produced
  - One for each reducer
  - Often uses a hash function or a range
  - Example:
    - md5(key) mod (#reducers)

Map Reduce in Detail

- **Reducer**
  - Shuffle and Sort
  - **Part of the map-reduce framework**
    - Incoming key-value pairs are sorted by key into one large data list
    - Groups keys together for easy agglomeration
    - Programmer can specify the comparator, but nothing else

Map Reduce in Detail

- **Reducer**
  - reduce
    - Written by programmer
    - Works on each key group
    - Data can be combined, filtered, aggregated
    - Output is prepared

Map Reduce in Detail

- **Reducer**
  - Output format
    - Formats final key-value pair

Map Reduce Patterns

- Summarizations
  - Input: A large data set that can be grouped according to various criteria
  - Output: A numerical summary
  - Example:
    - Calculate minimum, maximum, total of certain fields in documents in xml format ordered by user-id

Summarization

- Example:
  - Given a database in xml-document format

```
<row Id="193" PostTypeId="1" AcceptedAnswerId="194"
CreationDate="2010-10-23T20:08:39.740" Score="3" ViewCount="30"
Body="&lt;p&gt;Do you lose one point of reputation when you
down vote community wiki?  Meta? &lt;/p&gt;&#xA;&#xA;&lt;p&gt;I
know that you do for &quot;regular questions&quot;. &lt;/
p&gt;&#xA;" OwnerUserId="134"
LastActivityDate="2010-10-24T05:41:48.760" Title="Do you lose
one point of reputation when you down vote community wiki?
Meta?" Tags="&lt;discussion&gt;" AnswerCount="1"
CommentCount="0" />
```

  - Determine the earliest, latest, and number of posts for each user

Summarization

- Mapper:
  - Step 1: Preprocess document by extracting the user ID and the date of the post
  - Step 2: map:
    - User ID becomes the key.
    - Value stores the date twice in Java-date format and adds a long value of 1

```
“134”:  (2010-10-23T20:08:39.740, 2010-10-23T20:08:39.740, 1)
```

Summarization

- Mapper:
  - Step 3: Combiner
    - Take intermediate User-ID — value pairs
    - Combine the value pairs
      - Combination of two values:
        - first item is minimum of the dates
        - second item is maximum of the dates
        - third item is sum of third items

Summarization

- The map reduce framework is given the number of reducers
  - Autonomously maps combiner results to reducers
  - Each reducer gets key-value parts for a range of user-IDs grouped by user-ID

Summarization

- Reducer:
  - Passes through each group combining key-value pairs
  - End-result:
    - Key-value pair with key = user-id
    - Value is a triple with
      - minimum posting date
      - maximum posting date
      - number of posts

Summarization

- Reducer:
  - Each summary key— value pair is sent to client

Summarization

- Example (cont.)

Mapper 1

UserID 12345	01.02.2010	01.02.2010	1
UserID 12345	02.02.2010	02.02.2010	1
UserID 12345	04.02.2010	04.02.2010	1
UserID 98765	12.02.2010	12.02.2010	1
UserID 98765	02.02.2010	02.02.2010	1
UserID 98765	05.02.2010	05.02.2010	1
UserID 56565	02.02.2010	02.02.2010	1
UserID 56565	03.02.2010	03.02.2010	1

Combiner →

UserID 12345	01.02.2010	04.02.2010	3
UserID 98765	02.02.2010	12.02.2010	3
UserID 56565	02.02.2010	03.02.2010	2

Mapper 2

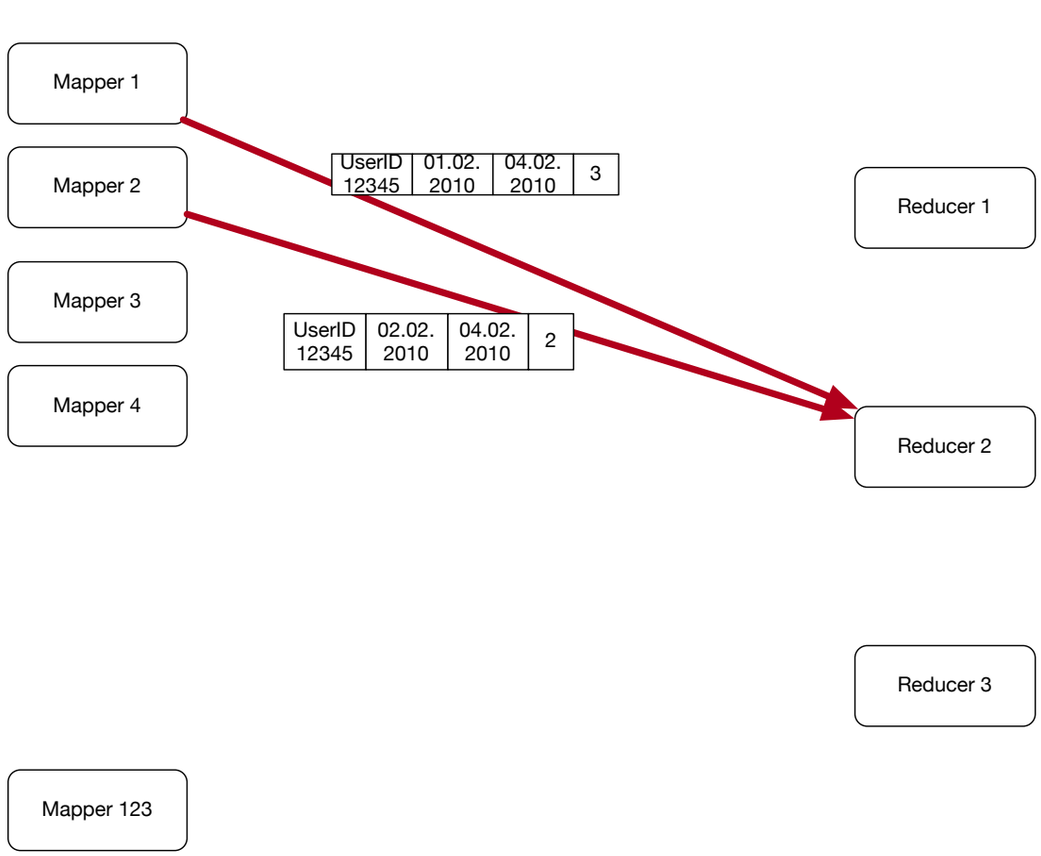
UserID 12345	02.02.2010	02.02.2010	1
UserID 12345	04.02.2010	04.02.2010	1
UserID 77444	12.02.2010	12.02.2010	1
UserID 77444	02.02.2010	02.02.2010	1
UserID 98765	05.02.2010	05.02.2010	1

Combiner →

UserID 12345	02.02.2010	04.02.2010	2
UserID 77444	02.02.2010	12.02.2010	2
UserID 98765	05.02.2010	05.02.2010	1

Summarization

- Example (cont.) Automatic Shuffle and Sort
  - Records with the same key are sent to the same reducer



Summarization

- Example (cont.)
  - Reducer receives records already ordered by user-ID
  - Combines records with same key

UserID 12345	01.02.2010	04.02.2010	3
UserID 12345	02.02.2010	04.02.2010	2
UserID 12345	26.03.2010	30.04.2010	5
UserID 12345	19.01.2010	01.04.2010	3
UserID 16542	02.02.2010	04.02.2010	6
UserID 16542	26.03.2010	29.05.2010	5
UserID 16542	19.01.2010	19.01.2010	1

UserID 12345	01.02.2010	30.02.2010	13

UserID 16542	19.01.2010	29.05.2010	12

Summarization

- In (pseudo-)pig:
  - Load data

```
posts = LOAD ‘/stackexchange/posts.tsv.gz'
USING PigStorage('\t') AS (
post_id : long,
user_id : int,
text : chararray,
…
post : date
)
```

Summarization

- In (pseudo-)pig:
  - Group by user-id

```
post_group = GROUP posts BY user_id;
```

- Obtain min, max, count:

```
result = FOREACH post_group GENERATE group,
MIN(posts.date), MAX(posts.date),
COUNT_STAR(post_group)
```

Summarization

- In (pseudo-)pig:
  - Load data

```
orders = LOAD ‘/stackexchange/posts.tsv.gz'
USING PigStorage('\t') AS (
post_id : long,
user_id : int,
text : chararray,
…
post : date
)
```

Summarization

- Your turn:
  - Calculate the average score per user
  - The score is kept in the “score”-field

Summarization

- Solution:
  - Need to aggregate sum of score and number of posts
  - Mapper: for each user-id, create a record with score

    `userid: score, 1`
  - Combiner adds scores and counts

    `userid: sum_score, count`
  - Reducer combines as well
  - Generates output key-value pair and sends it to the user
  - `userid: sum_score/count`

Summarization

- Finding the median of a numerical variable
  - Mapper aggregates all values in a list
  - Reducer aggregates all values in a list
    - Reducer then determines median of the list
- Can easily run into memory problems

Summarization

- Median calculation:
  - Can compress lists by using counts
    - `2, 3, 3, 3, 2, 4, 5, 2, 1, 2` becomes

      `(1,1), (2,4), (3,3), (4,1) (5,1)`

  - Combiner creates compressed lists
  - Reducer code directly calculates median
    - An instance where combiner and reducer use different code

Summarization

Summarization

Summarization

Summarization

Summarization

Summarization

Summarization

Summarization

Map Reduce Patterns

Map Reduce Patterns

Map Reduce Patterns