Big Data Meets Statistics

...

Dr. Yijiang John Li

March, 2016

At the beginning ...

- About me
  - Michigan
  - Google
  - Caspida (now Splunk)
  - yijiang1121@gmail (or LinkedIn)

- Acknowledgement
  - Kathy and Oliver
  - Mentors, colleagues and friends

Road map

- Introduction
- Batch mode
- Streaming
- Lambda Architecture

Introduction

- Nowadays almost anything data related is tied to the term Data Science
  - Two types of data scientist: type A and type B (due to Michael Hochster)
  - Type A
    - A for Analysis; making sense of data
    - Strong statistical background; practical details of working with data that aren't taught in the statistics curriculum; communicating well about data
  - Type B
    - B for Building; building models which interact with users
    - Strong programming background; interested in using data "in production"
- Goal: to share my thoughts and lessons learned (focusing on B).

Introduction – what are models for

- The interpretation / discovery side
  - Often this involves the estimation of certain parameters of interest
- The prediction side
  - Batch mode
    - Can be used online as well
    - Infrequent model updates
  - Streaming mode
    - Frequent model updating (model aging)
    - Forgetfulness

Road map

- Introduction
- **Batch mode**
- Streaming
- Lambda Architecture

Batch mode – data at rest

- Start with a straightforward problem
  - Many domains (hosts)
  - Many features: global popularity and local popularity; reputation score; # paths; randomness ...
  - Interested in comparing domains w.r.t certain features
- Using percentile is very meaningful for comparison purpose
  - Also very useful if we want to flag a set of domains for further investigation
- Handling ties
  - Use average ranks

Batch – easy part

- Not a problem if data is relatively small and lives in a “siloed” environment
  - Easily fit in the memory of a single machine
  - Think about R data frame
  - R: rank(x, ties.method = “average”)
- This is quite common for one-off analysis
  - Such aggregated data are pulled from some data repo
  - Create an analysis report for business intelligence and etc

Batch – hard part

- What if data is reasonably big
  - could be millions of domains (even more if including paths) and hundreds of features
  - Interested in more granular levels of information, such as daily, weekly, monthly
  - Cannot be handled by a single machine
- What if data is no longer in silo
  - The input is from an upstream data pipeline
  - Results need to be made available to other components of the production environment
  - Close Integration with other components / services

Bridge this gap

- Know a bit more about the implementation details
  - Traditional statistical training: theory, applied methods
  - Computational statistics (+1)
  - Not to get lost in all the algorithm/programming details
- A working knowledge of distributive computing and big data
  - MapReduce paradigm
  - Apache Spark
  - Not really a difficult concept
  - Accumulated over time

Implementation details

- Lots of handy R functions / libraries are not available in other environments
- Some are only partially available
- Need customization
- Improve understanding
- Not really to reinvent the wheels

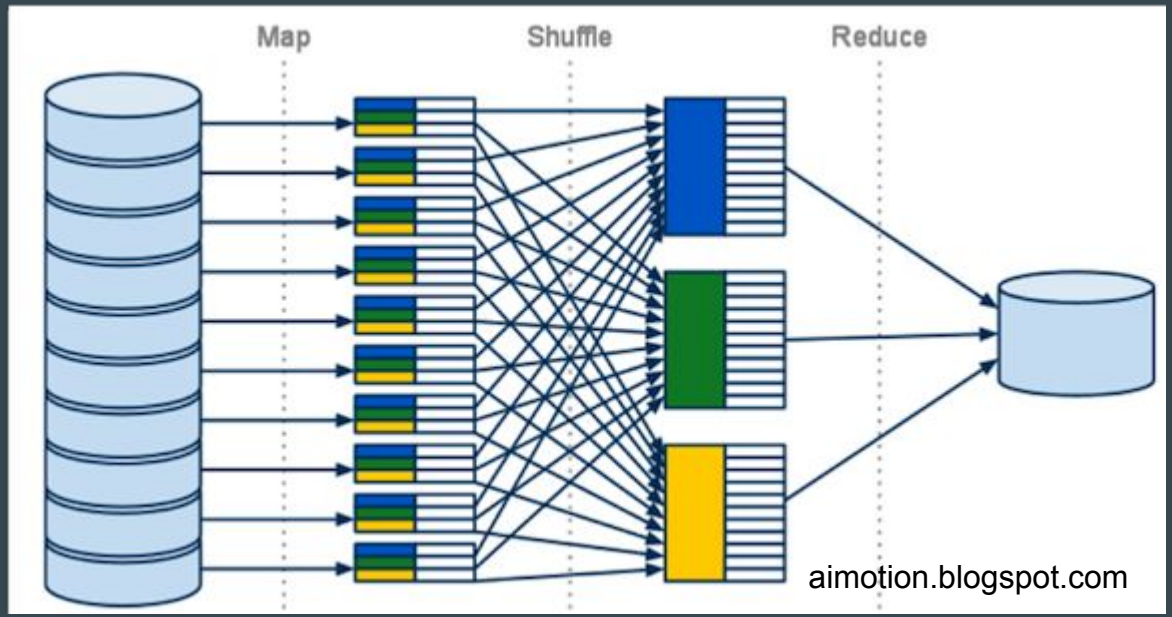
Implementation details

- Back to domain ranking example
  - Think about how rank is computed
  - Handling ties: the default method in Spark is not sufficient
  - Two approaches: functional vs imperative
- When data is big
  - Think about these implementation details in the context of distributive computing
  - Avoid potential pitfalls

Distributed computing – very brief intro

- MapReduce with an example
  - Huge log files stored in HDFS; want to compute domain (possibly including path) popularity
  - Could be that the end results -- all (domain, count) pairs -- cannot fit in memory
- Map
  - Each line -> (extracted domain, 1)
- Shuffle
  - Essentially group by key (domain in this case)
- Reduce
  - Aggregation, summation etc
  - (d1, 1), (d1, 1), (d1, 1) -> (d1, 3)

MR diagram



Not totally new to us statisticians

- We do things similar in R all the time

```
> data <- list("google.com", "amstat.org", "umich.edu", "umich.edu")
> str(data)
List of 4
 $ : chr "google.com"
 $ : chr "amstat.org"
 $ : chr "umich.edu"
 $ : chr "umich.edu"
```

Map

```
> map <- function(d) list(d, 1)
> after_map <- lapply(data, map)
> str(after_map)
List of 4
 $ :List of 2
  ..$ : chr "google.com"
  ..$ : num 1
 $ :List of 2
  ..$ : chr "amstat.org"
  ..$ : num 1
 $ :List of 2
  ..$ : chr "umich.edu"
  ..$ : num 1
 $ :List of 2
  ..$ : chr "umich.edu"
  ..$ : num 1
```

Shuffle

```
> after_shuffle <- split(after_map, unlist(data))
> str(after_shuffle)
List of 3
 $ amstat.org:List of 1
  ..$ :List of 2
  .. ..$ : chr "amstat.org"
  .. ..$ : num 1
 $ google.com:List of 1
  ..$ :List of 2
  .. ..$ : chr "google.com"
  .. ..$ : num 1
 $ umich.edu :List of 2
  ..$ :List of 2
  .. ..$ : chr "umich.edu"
  .. ..$ : num 1
  ..$ :List of 2
  .. ..$ : chr "umich.edu"
  .. ..$ : num 1
```

Reduce

```
> reduce <- function(list) sum(sapply(list, function(each) each[[2]]))
> after_reduce <- lapply(after_shuffle, reduce)
> str(after_reduce)
List of 3
 $ amstat.org: num 1
 $ google.com: num 1
 $ umich.edu : num 2
```

Spark and RDD – very brief intro

- RDD: Resilient Distributed Dataset
- In memory > on disk > over the network
- Back to domain ranking example context
  - Avoid pitfalls: groupByKey, reduceByKey, sortByKey

...	...
d2	18.5
d3	18.5
...	.

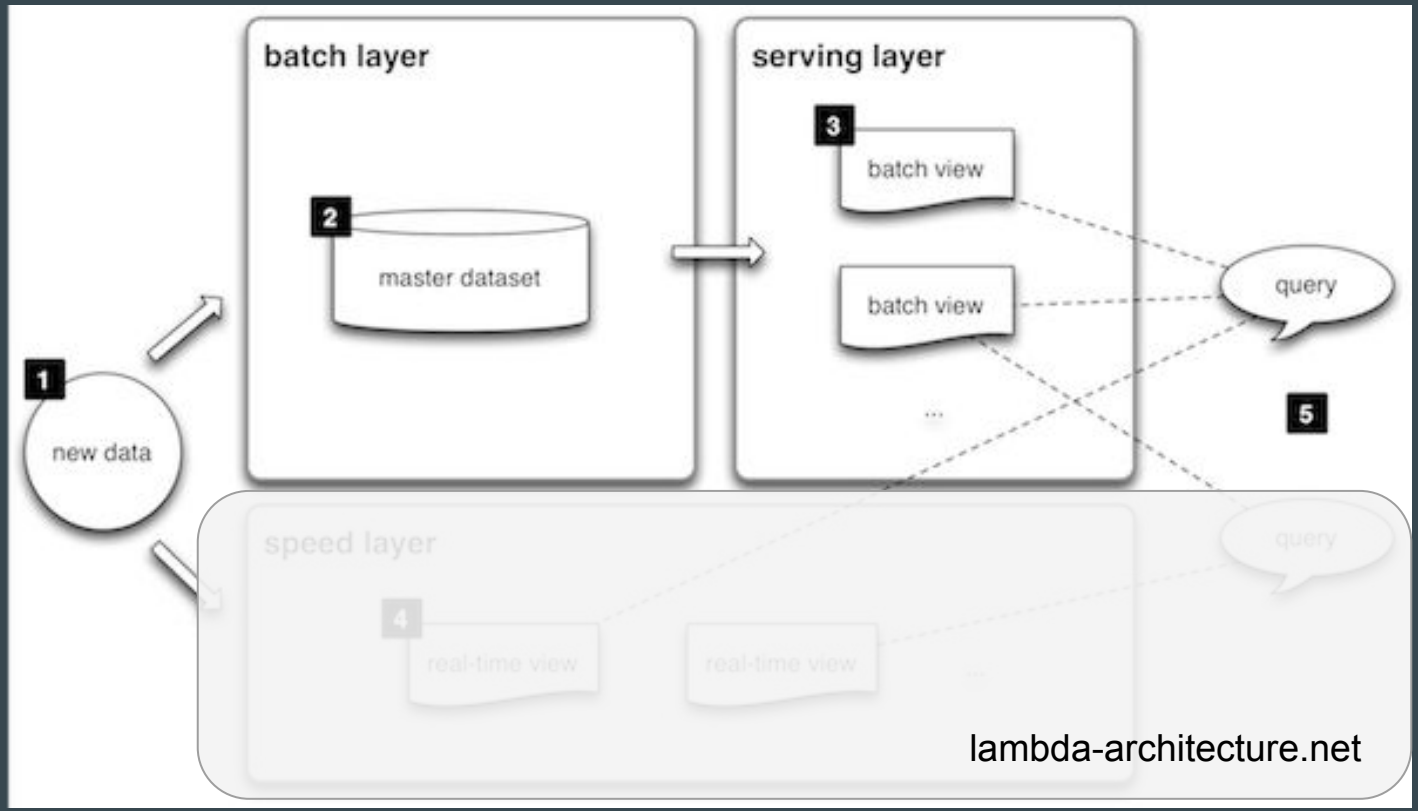
Sort by value;
Take index

d1	18.5	10
d2	18.5	11
d3	18.5	12
d4	17	9

Reduce (several choices)

d1	18.5	11
d2	18.5	11
d3	18.5	11
d4	17	9

Lambda architecture – batch view

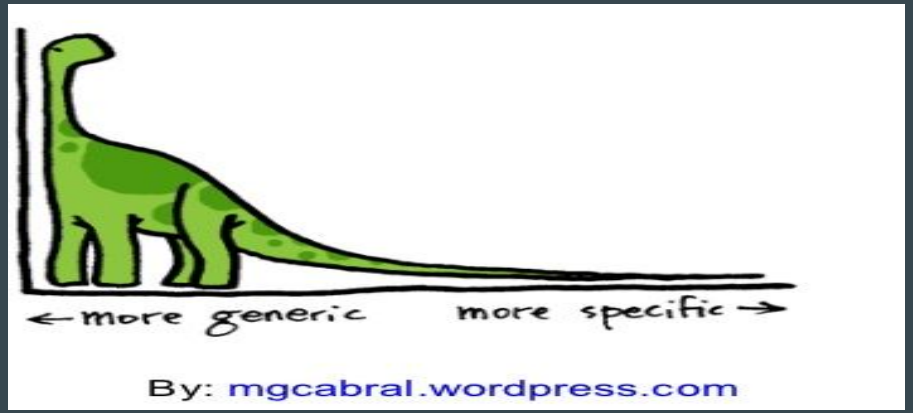


Road map

- Introduction
- Batch mode
- **Streaming**
- Lambda Architecture

Example: rare levels of categorical variable

- Problem of interest
  - To find levels (if any) of a categorical variable that occur rarely
- One straightforward solution
  - To estimate the probability that each level occurs
- Issue
  - Long tail (i.e. high cardinality) variables
- How to address this?



Example: rare levels

- Borrow the p-value concept
  - Previously: estimating one probability for each level
  - Now: estimating sum of probabilities for each level
- The same binomial formulation
  - If needed, we could use upper confidence bound to be on the conservative side
  - For each level, we can define its rarity score
- Straightforward and easy to implement
  - +1

Example: rare levels

- So far we assumed data at rest
- Batch processing
- What if data arrives in streaming mode? Or your wanna near real-time detection
    - Fast data

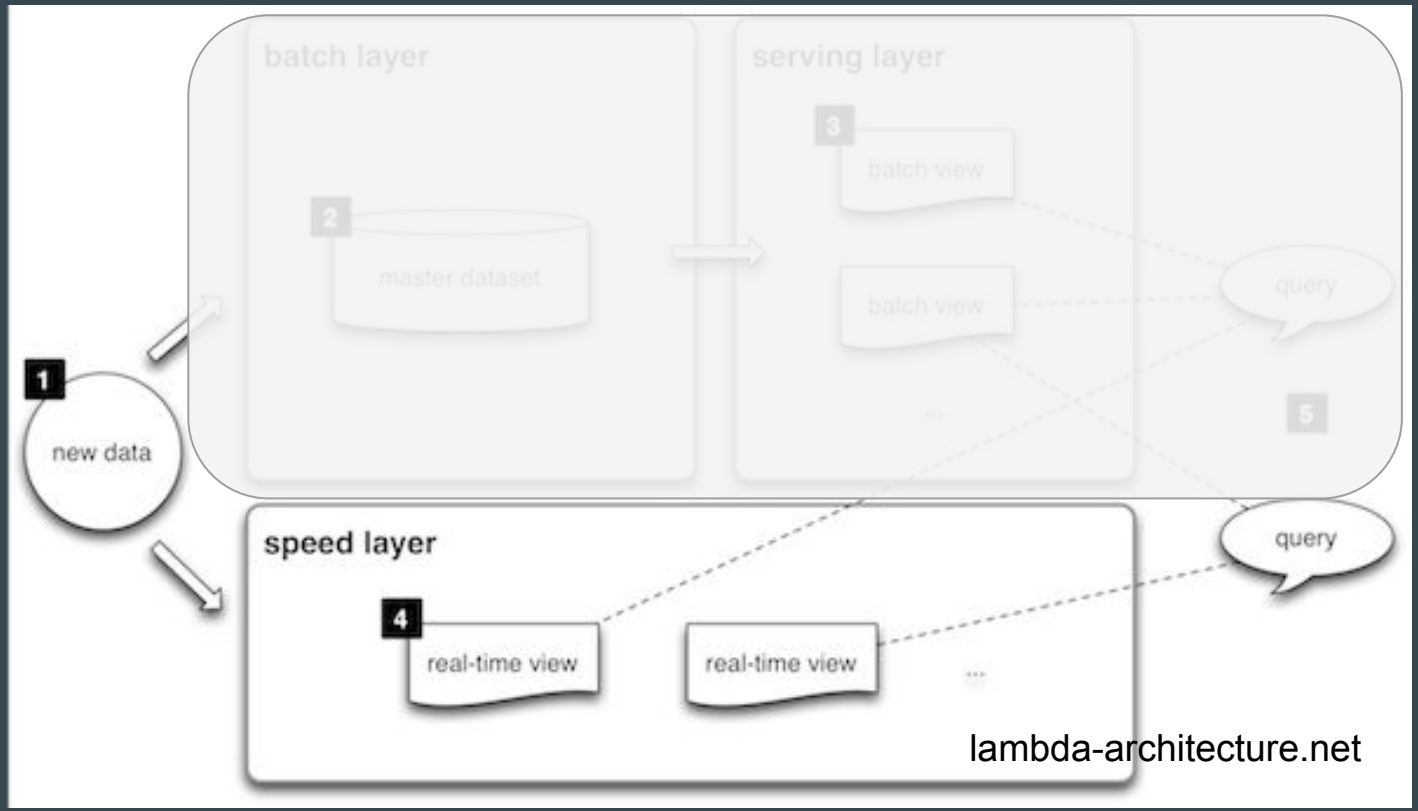
Rare levels - streaming

- HashMap like data structure for data representation
  - Map(level -> count)
- First compute and then update for each arrival event
  - Compute: lookup and scan/filter operations
  - Update: insertion operation
  - Could do this in reverse order as well
- This is fine if data arrive at a slow/moderate rate
- What if data arrived very fast?

Rare levels - streaming

- The problem with the initial computation logic
  - A whole map scan/filter operation for each arrival event
- Need additional data structure
  - Red-black tree among other possible choices
  - Quickly find levels that are less frequent than the current one
- Additional time needed for the updating logic
  - Update this new data structure after computing for each event
- What if data arrived REALLY fast
  - Come back later

Lambda architecture - streaming view



Rare levels – summary

- Statistical aspects
  - p-value like idea (conservative in practice)
  - focusing on computing confidence interval
- Engineering aspects
  - Speed in practice
  - Implementation details

What if data arrives REALLY fast

- Consider Approximation
  - Generally applicable; often times a good idea
- Events -> model (which keeps updating itself) -> output
  - Each component can be viewed as a trade-off between accuracy and computational performance
  - From single event to small batch of events
  - From model updating per event to updating per bulk of events
  - Heuristics for computing the output
- Other than approximation, any systematic approach/framework?
  - Lambda Architecture

Road map

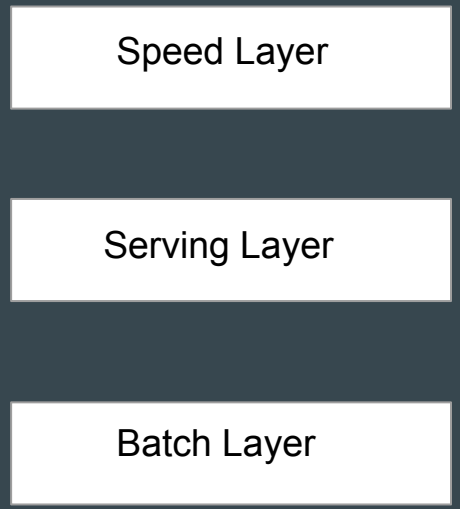
- Introduction
- Batch mode
- Streaming
- **Lambda Architecture**

Back to this “rare levels” example

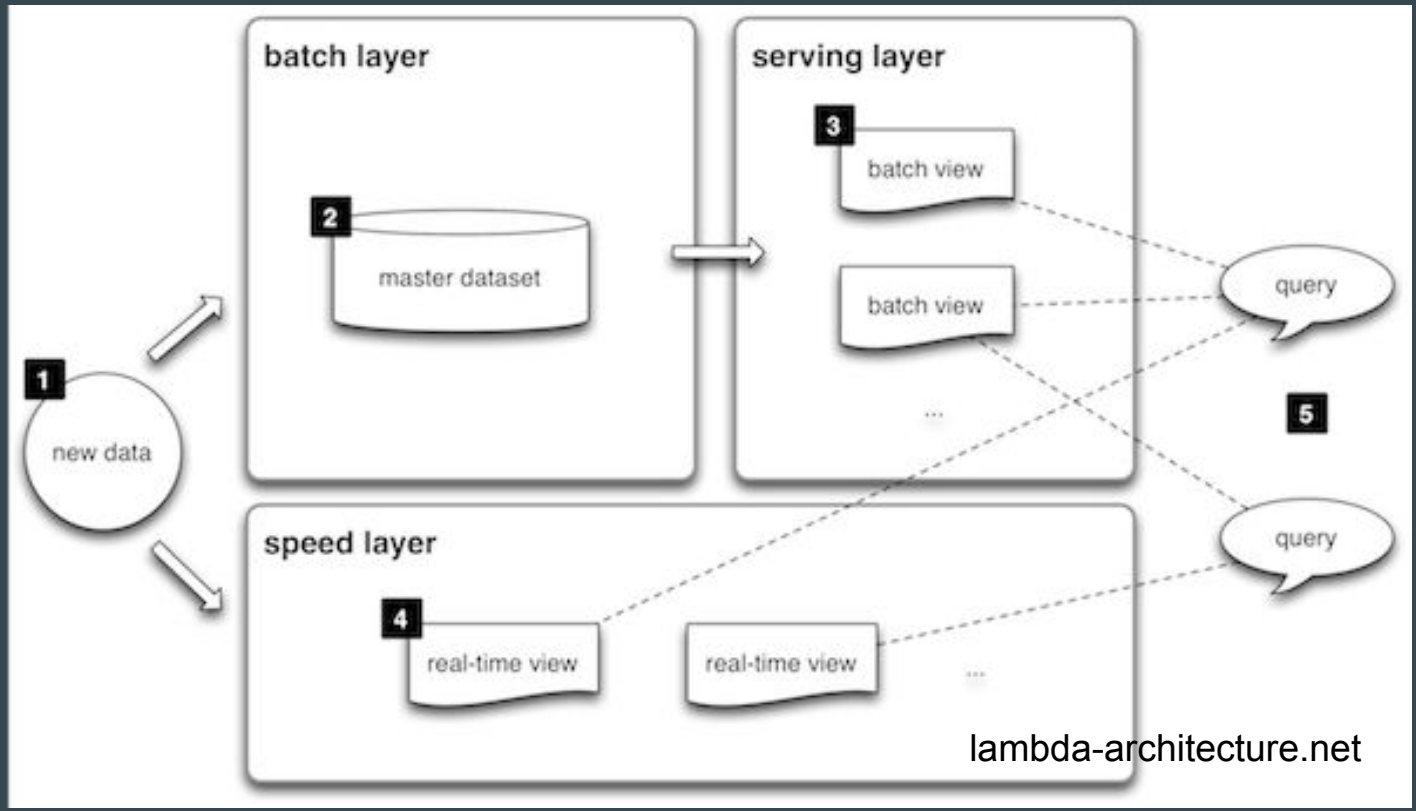
- What if data arrived really really fast
  - HashMap time complexity: essentially constant
  - This additional red-black tree: log(N)
  - The streaming model will be inevitably slow after a while (especially for fast data)
- Batch models take much less time **per event** to build
  - But it’s NOT kept up-to-date
- Enjoy the best of both worlds
  - Lambda Architecture

Lambda Architecture – scratching the surface

- Due to Nathan Marz
- *Query = function(all data)*
  - Not feasible
- Essential idea
  - *Batch view = function(all data)*
  - *Realtime view = function(Realtime view, new data)*
  - *Query = function(Batch view, Realtime view)*
- Batch layer scales by just adding new machines



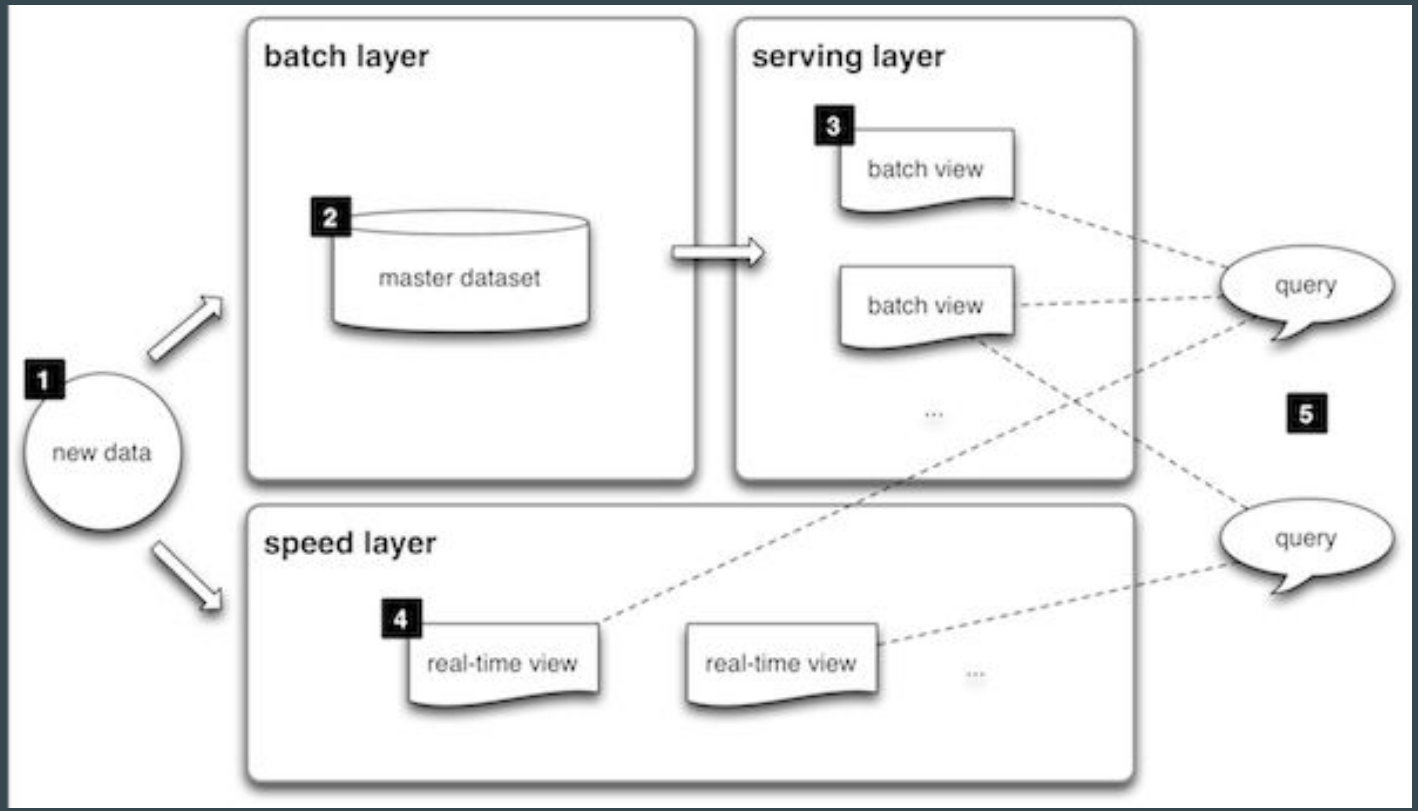
Lambda Architecture



Kafka and Cassandra (very briefly)

- Kafka
  - Originally from LinkedIn
  - Distributed publish-subscribe messaging system
  - Producer, Consumer and Broker
- Cassandra
  - Originally from Facebook
  - Distributed feature from Amazon Dynamo
  - Data model from Google BigTable

Kafka and C* in Lambda Architecture



Thank you!

Questions