Spark in production pipelines

EXPERIENCES FROM AMAZON FORECASTING

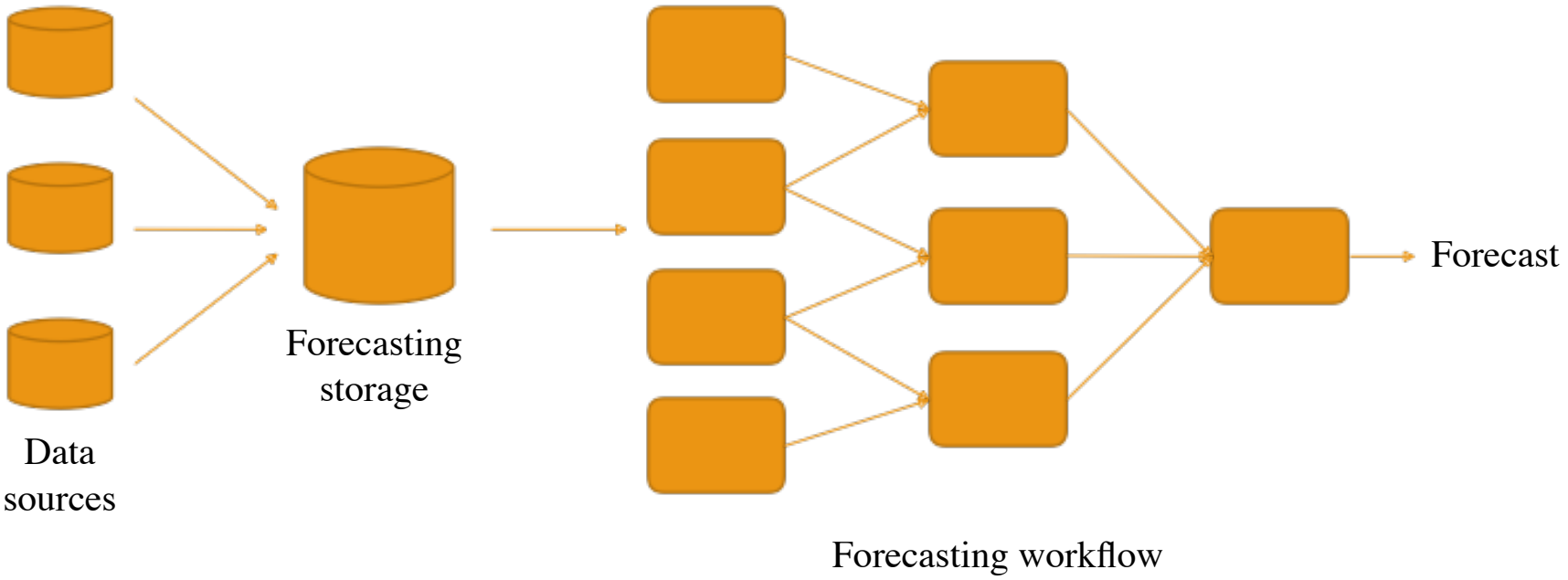
Amazon Forecasting

- We forecast demand for products we sell
- Fascinating machine learning problem
- Use any data we can get our hands on
- Constantly experiment with new methods of forecasting

Amazon Environment

- Two-Pizza Teams
- SOA – services
  - We hate code dependencies
- Oncall rotation
- Lots of data
  - 20PB of data in archive
  - 1PB of data (parquet, compressed) used in Forecasting

Forecasting Environment



Forecasting Environment

- A mix of legacy, current and future applications
  - Constantly developing infrastructure
  - Hadoop / HDFS Legacy applications
  - Cannot assume uniform, homogeneous environment
- Spark on EMR is being used in new applications
- Hadoop on permanent shared cluster is being used in legacy applications
- Permanent data on S3, consumed directly
- Complicated workflow with many stages, culminating in creation of the forecast
- Separate team is tasked with data acquisition

Big data is hard

- Most of the problems are solved elsewhere, but
  - …not at the same time!

- No pre-existing knowledge base, industry standards, best practices
  - RDBMS technology exists since 1970's
  - MapReduce exists since 2004
  - Spark exists since 2012
  - Spark as a production tool?...

- These new technologies must be made stable enough to serve Amazon production

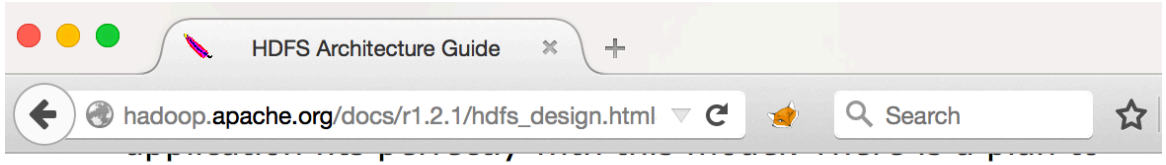
Wrapping Spark

- Do not do it!
- We have wrapped Hadoop. Did it help for Spark migration?
- Implement transformations as sharable components exchanging Spark primitives

Dataset concepts

- Datasets and not files!
  - ”I have a dataset stored in Parquet file…”
  - “My dataset is already in a CSV file…”
- Datasets are not directories, either!
- Logical entities, with some similarity to *tables*, but also some differences.
  - Partitioned
  - Accessible
  - Versioned
  - Immutable
  - Durable

Store / Compute Separation



support appending-writes to files in the future.

"Moving Computation is Cheaper than Moving Data"

A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

Store / Compute Separation

- Moving computation is *way* more expensive than moving data
  - Computation requires permissions
  - Computation produces dependencies
  - Computation requires compatibility
- Data storage makes hardware stateful… and precious!
  - Cannot swap out one cluster for another
  - Cannot tailor HW type for job profile – CPU bound, memory bound etc.
  - Must keep HW up 24/7
  - Loss of HW implies loss of data
- Moving data is actually cheap
  - S3 / EMR pairing works really well
  - Permissions are much simpler to set up

Delay Materialization

- Applying transformations to data prior to writing
  - Aggregation
  - Removing fields
  - Removing records (filtering)
- Scenario: business logic requirements change
  - New aggregation is required (weekly instead of monthly)
  - New field is required (and we've already thrown it away)
  - Different filtering is required
- Result: BACKFILL
  - Original data may or may not be available
  - Schema is modified
  - Customers are asked to migrate

Delay Materialization

- To the extent possible, transformations to data should be made on READ
- Caching can be used to avoid repeated computations
  - Transparent to the calling application
  - Original data is preserved, so all computations can be repeated
- Logic should be shared as code or service, not as data

Delay Materialization - examples

- Data source contains sensitive information – REMOVE
- Data source is disaggregated, but your customer needs weekly aggregation – KEEP
- Data source has inconsistent province names – Ontario, Ont., ON, etc. - ???
  - Save raw source and modified dataset – but what about streaming?
  - Will anyone ever want to undo this?

Delay Materialization – sharing code

- Library: but does everyone use Spark?
- Binary: but does everyone has compatible environment?
- SQL registry: but are SQL dialects really compatible? And does everyone know how to apply SQL?
- Service: but can you throttle / schedule big data jobs under strict SLA?
- Call-back service: but is everyone's HW compatible?
- Spin-up/Spin-down service ("bring your own account"): performance
- Record-by-record SOA: performance, high frequency service

Metadata is… big data

- We need to process many thousands of Parquet files at once
- Default behaviour wants to open each file
- That's 4 fstat / seek operations!

- Parquet's solution: directory manifest
  - But… datasets are not directories!
  - S3 hashing

Big metadata, take 1: file per split

- Count the files
- If too many, avoid reading manifest on master node: task.side.metadata = true
- Also avoid getting file size (hacked Parquet code)
- One file per Hadoop Split / RDD partition

- This doesn't work very well
  - How many is too many? Depends on cluster size
  - For some cluster sizes, you either have too few partitions or too lengthy processing on master
  - No control over partition size

Big metadata, take 2: use Spark tools

- Use Spark to compute RDD parts by distributing file access
- Parallelize a list of paths, send to each node (rdd.map())
- Node gets file size, sends back (rdd.collect())
- Split files into partitions on Master

- …this is better
- Still using task-side metadata, so partitions are not aligned to pages

Test at Correct Scale

- System tested at TB scale will not perform the same at PB scale
- Example: Parquet Integer Overflow
  - Nested schema: MAP containing ARRAYs
  - > 2 billion elements in column
  - Integer overflow
  - PARQUET-511

Conclusions

- Do not wrap spark
- Define a dataset as logical entity
- Maintain store / compute separation
- Metadata can be Big Data
- Do not modify data on WRITE (if you can)
- Test at Correct Scale