Shoehorning Spark:

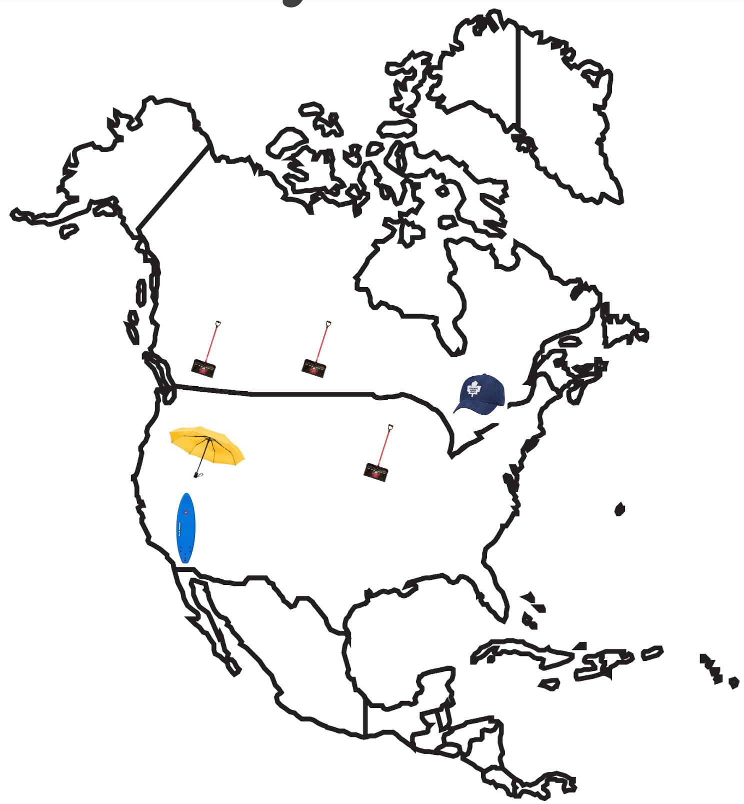
DRAGGING A LEGACY WORKFLOW SYSTEM INTO THE 21ST CENTURY

SEBASTIAN KUN, SR. SOFTWARE DEV ENGINEER

AMAZON.COM

Our problem space: Inventory Placement

- For every retail item Amazon carries, decide what % to put in each warehouse
- Put things where we think people will order them in the future
- Constrained optimization problem:
  - Minimize fulfillment costs
  - Obey capacity limits at each warehouse

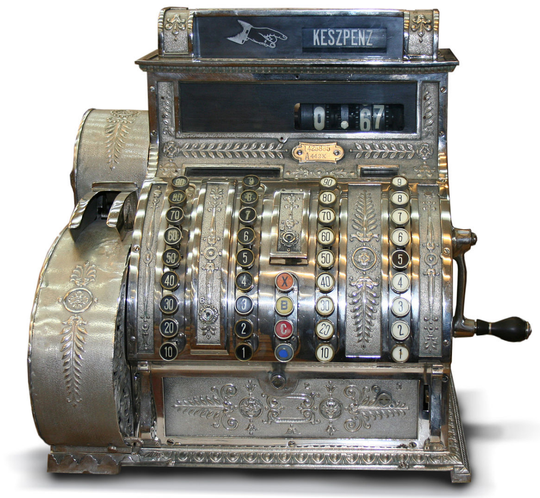


Our problem space: challenges

- Displacement: most demand comes from big cities, but that where the smallest warehouses are
- Seasonality: yearly demand shifts (textbooks, air conditioners, turkey fryers)
- Multi orders: disadvantages smaller warehouses with less selection
- Local availability: can’t fulfill same-day across the country
- Different capacity constraints: labour, cubic storage, truck space
- Different warehouse specializations: big/small items, hazmat, cold storage

Our legacy system

- Daily datasets: tab-separated text (TSV) with header, stored on NFS
- Job: read input, process using a Java class, write output
- Job graph: jobs/datasets are vertices, input/output dependencies are edges
- Fleet of worker hosts, take jobs from a central queue
- Most datasets are < 5GB/day, not that big



What do these jobs do?

- Load external inputs: data warehouse extracts, service calls, S3 downloads
- Manipulate data: join, filter, aggregate, sort
- Validate according to business rules
- Mathematical optimizations: linear programming, min cost flow, …
- Publish data externally:
  - Automated buying systems
  - Analysts, researchers, operations team

Throw everything into a database?

- Is this just an ETL system? Should we put all this into a data warehouse system?
- TSV file is a lot like a DB table, many operations are very SQL-like
- Some things can be expressed more concisely with SQL
  - Beware the 700 line SQL Extract Of Death
- We like to have the flexibility to have different programming models, whatever solves the problem best
- System works “well enough”, hard to build up the business case to overhaul

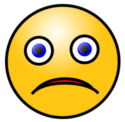
The business case: a new inventory placement algorithm

- One step would produce an intermediate data set with a row for every combination of: item, warehouse, geographic region, shipping option
  - Trillions of combinations
  - Also needs to be sorted
- Again: throw everything into a database?
  - Redshift: AWS data-warehouse-on-demand, scales to PB of data
  - Still limited to SQL interaction model
- What about Spark?
  - Just set the 100TB sorting record
  - Flexible; could implement other parts of the algorithm, not just the sorting

Early experimentation

- Very easy to get started with Spark-shell + standalone mode
- Wrote a proof-of-concept in 80 lines of (naïve) Scala
- Included not just the sorting and TSV-handling code, but generated the data set from source files
- 10x speedup vs. earlier prototype code + UNIX sort (single host, subset of data)
- Convinced us it was worth spending more time on

Integrating into our existing system

- Existing system used Java 7
- Java 7 + Spark: 
- Java 8 + Lombok + Spark: not bad
- Reuse our existing workflow system for orchestration
- Start on small datasets using Spark in standalone mode – can run on existing workers
- Later enhancements: cluster on AWS EMR, use binary file format (Parquet), store files on S3

Benefits for analysts

- Ad-hoc queries on TSV files are a big pain!
- E.g. given two datasets:
  sales_data [itemID, qty] item_attributes [itemID, product_line, size_bucket]
"how much are we selling in total, in each product line+size bucket?"

UNIX:
```
join -t $'\t' <(zcat item_attributes.tsv.gz | sed -e 1d | sort) \
             <(zcat item_sales.tsv.gz | sed -e 1d | sort) \
 | awk -F $'\t' 'BEGIN {OFS = FS;} {sum[$2 "\t" $3] += $4;} END {for
(plSize in sum) {print plSize, sum[plSize];}}' \
 > output.tsv
```

SQL:
```
SELECT attrs.product_line, attrs.size_bucket, SUM(sales.qty)
FROM  attrs JOIN sales ON (attrs.itemID = sales.itemID)
GROUP BY attrs.product_line, attrs.size_bucket
```

Benefits for analysts

- Exposed SparkSQL functionality using a command line tool
- Later: nicer web tool with autocomplete, syntax checking
  - Eventually add notebook functionality
- Even analysts don't like having to use SQL for everything
  - Avoid hacks like segment tables, 'select X from dual union…', etc.
  - Sometimes a little Python goes a long way
- Storing data on S3 is way cheaper than storing on a DB cluster
  - Can do year-over-year analysis on more datasets
  - Keep a long tail of rarely-used data

Overall lesson

- …is not that Spark is the best or that we use Spark for everything
- Don’t be dogmatic, seeing your tool as a hammer and every problem as a nail
  - Analyst: everything is SQL!
  - Data Scientist: everything is machine learning!
  - Operations Research Scientist: everything is a linear program!
- Choose the best tool for the job
- Spark gave us a lot of flexibility around API (RDD, Dataframe, SQL), data types (columnar, semi-structured) and storage (file, S3, HDFS)
- Flexibility benefitted not just engineers but also analysts, researchers
- Useful even for smaller datasets