The Continuing Story Of Analytics at Optimizely:

Batch, Streaming and Lambda Systems

Mike Borsuk

mike.borsuk@optimizely.com
About Optimizely

Experiment Everywhere
  o Experimentation, Personalization, Recommendations
  o Web, Mobile, OTT, Full stack

Data challenges
  o Billions of events per day received
  o Real-time results
Overview

- Background & Motivation
- Real Time Stream Processing
- What is Lambda Architecture and how/why we are implementing
Optimizely X Personalization

Overview of Results for Homepage Promotions

- PRIMARY EVENT: Overall Revenue
- OVERVIEW: Homepage Promotions +1.4%
- AUDIENCE:
  - Warm Weather (via Skymosity) +8.4%
  - 18.9% of all traffic $16,395
  - Bag Shoppers -1.3%
  - 15.9% of all traffic -$3,248

- TOTAL IMPROVEMENT IN OVERALL REVENUE: +1.4%
- TOTAL INCREASE IN OVERALL REVENUE: $13,147
- TOTAL INCREASE IN OVERALL REVENUE PER VISIT: $0.31
- TOTAL VISITS: 35,192
- TOTAL REACH: 34.8%
- DAYS RUNNING: 4
- LAST PUBLISH: June 2015
- DATE CREATED: June 2015

Increase in Overall Revenue Over Time

Reach

- Warm Weather
- Bag Shoppers
- Everyone Else

Percent of All Traffic: 58.6%

Optimizely
Personalization data scale

- 4.14B raw events received daily
- Grouped into 10M distinct visitor sessions daily (stream processing w/Samza)
- Calculating and serving back millions of time series data points
Personalization data challenges

- From a single A/B test per experiment to multiple targeted tests in a campaign
- Longer running data collection / analysis
- Need for session based metrics
- Data schema designed for single A/B tests
Personalization data scale

- **Mean** response time (HBase) goes from milliseconds to nearly 30s
Realtime Stream Processing

Persist raw events
  - S3 buckets grouped by 24h UTC

Fan out events into processing queues
  - Kafka topics for event types

Session aggregation w/Samza
  - Groups clickstream events into sessions
  - Per-visitor basis
  - Split on 30 minutes inactivity
Stream Processing Architecture

- Event APIs
- Kafka
- Visitor Partitioning
- Session Aggregation
- Samza

Stream Processing

Storage

Serving

- S3
- HBase
- DynamoDB (Cache)
- Serving APIs
- Time Series Computation & Aggregation
Lambda Architecture

- Batch Layer
- Serving Layer
- Speed Layer
Lambda Architecture
Our Implementation of LA

- Match schema to query patterns
- Make time-series data “combinable” or at the same base granularity
- Write data into HBase for locality at query time, “de-normalization”
Our Implementation of LA

- Immutable raw-event “source of truth”
- Pre-computation batch jobs matching our real-time
- Time range optimized real-time queries
- Serving layer to merge batch + real-time
- Done for performance, not accuracy
Adding Lambda Layers
Adding Lambda Layers

- **Pre-computed Time Series**
  - Batch Layer
  - Serving Layer
  - Realtime Computation

**Composite Time Series Result**

**query time range**
Benefits we are seeing

- Solving our query latency issues
Benefits we are seeing

- Flexibility
- System Fault Tolerance
- Human Fault Tolerance
Drawbacks we are seeing

- Complexity in serving layer
- Batch job management
- Operational Burdens
References

Big Data, book by Nathan Marz and James Warren

Optimizely engineering blog: https://medium.com/engineers-optimizely

Samza specific: Optimizely presentation at LinkedIn streaming meetup (https://youtu.be/p7hjrKyfQkc)