The world wide 60 billion transaction per day journey

We

- Ranbir Chawla
  - V.P Engingeering
  - Ranbir.Chawla@audiencescience.com
- Frank Conrad
  - Chief Architect
  - Frank.Conrad@audiencescience.com

Background

- AudienceScience provides fully integrated, end to end, advertising solutions for the world's largest brand advertisers.
- AudienceScience receives, processes and responds (in real-time) to over 80 billion incoming requests a day, in over 42 countries.
- Our solutions allow advertisers to effectively manage and leverage their consumer data to produce industry leading ROI on their advertising spend.
- Global Distribution of Five Points of Presence to Central DC
- Where we were
  - 20 Billion TPD in 2014

The Challenges of Scale

- Scaling is in the details
- We often miss the obvious steps in looking @ the complex problem
- Clever design does not solve for bad execution

Chose the right data model

- What scale you need – allow for parallelism
- Store the data that you effective can use it
- How it can handle out layers
- How add later changes, it is extensible
- How is data expired / deleted

Chose production oriented architecture

- To allow scale
  - With parallel (as massive as make sense)
  - Dynamic parallelisms
  - Asynchronous processing
  - Look to latency and throughput
  - Eventual consistency is possible
- Production oriented
  - Dynamic limiting
  - Allow catch up
  - How to handle unreliable network
  - Uneven / unreliable hardware
  - Think about out layers (latency, data size, cpu consumption), think was to do with them
  - Monitor, monitor, monitor

AUDIENCE SCIENCE

Monitoring – How are you scaling today?

- Chose a monitoring tool set that you can easily script and version control
  - We leverage nagios in our environment
  - All setup and configuration happens in an automated fashion
  - Bad hardware in large distributed clusters can kill an entire workflow
- Functional Monitoring
  - End to End monitoring – injecting known data for known result
  - each component in the pipeline plays a part here
  - Monitor all of your instances – sampling does not work here

Hadoop at the core

- Our main computational engine is still Hadoop
- Started with 60 Nodes, now running > 500 Nodes
- Leverage best practices to scale Map Reduce
  - As much compute as possible in the mapping phase
  - Minimize shuffle data, leverage locality
  - Output files must be in optimal format, sizes etc for the consumers next in the workflow
  - Put not all nodes in one cluster, have 2 or even more
    - But let them easy to move between
    - No single point of failure, simpler update

Hadoop Tuning

- Optimize your job in balance between:
  - Mapper/Reducer runtime (good is 5min)
  - Number of mapper / reducers, have shuffle time under control
  - Amount shuffle data
  - Amount and size of output files
- Focus on not creating extra garbage monitoring the GC is difficult
- Make sure that JVM setting is always in UTF-8
- Enable speculative task execution (but not for S3 writes)
- tmp space distribute across all drives
- Create filesystem instance for each data bucket

Leveraging Storm

- Only use grouping if you really need it
- If processing time for certain inputs have a large distribution Storm will have issues. Work to get consistency
- minimize cross JVM/Node traffic == minimize shuffling again!
- As always not create extra garbage, keep the GC happy

Voldemort as a large scale key store

- Memory only based stores scale to large sizes
  - Very Stable
  - scales well and is performant
  - But no monitoring what is inside memory
- read only stores
  - Efficient to produce with MR on scale
  - scales well and is performant
  - Very stable
- Use newest Voldemort client 1.10, handle failure / edge cases better

Voldemort Continued

- Challenge is deploying huge RO stores from MR to Voldemort cluster
  – Huge impact to page cache and disk IO on download new one
  – Internal solution
    - work with FS and HDFS only
    - not good to control
    - Failure handling, recovery is difficult
  – We use our own solution
    - Leverage cloud

Kafka to feed data from around the world

- Clusters
  - POD 5 x 6 broker
  - DC 16 broker
- Hardware
  - Large disk capacity per broker migrating to smaller capacity more brokers
- Production
  - Stable (use 0.8.1.1, plan to migrate to 0.9)
  - scales well and is performant
  - Mostly hardware and human related issues
  - Older version have high load to zookeeper
- Learning
  - 17 TB of data is to much per server, in terms of failure recovery
  - Mirror Maker scale need a lot of tuning
    - Compression cost huge CPU
    - A lot of instances/parallel connection to get throughput with latency

Scaling Cassandra

- Cross Data Center replication needs solid networking and very focused deployment
  - Repairs become a challenge
- Optimize your data model to avoid deletes
- Leverage native C* TTL as much as possible
- C* works will for ‘time-series’ oriented data, leverage time aspects of your data model – TTL again
- In the end avoid un-necessary clean up jobs
- Key/Value mapping if the value is of similar size to the key C* is inefficient and needs specific tuning.
- On SSD, make sure TRIM works

Scaling Clusters and Micro-Services

- We use Mesos/Marathon on Bare Metal and in AWS
- Deploy with gradual scale vs. single massive deployment
- Local docker registry
  - Need good network, IO, to deploy fast
- Cleanup
  - Unused images
  - Old instance data
  - Runtime log files
- Minimize Docker images, but keep them debugable (can install tools on demand)

Moving to the Cloud

- Focused on handling variable, ‘elastic’ data
- Challenges when moving off of fixed hardware onto someone else’s network
- Object Store vs. HDFS
  - If using Object Stores use them only @ the start and end
  - Hash S3 Object Store data for efficient writing
  - HDFS for temporary storage
  - Consider HDFS full time and add ‘compute only nodes’ to scale up
- Leveraging Qubole for scalable Hadoop/Spark

Scaling Time Series oriented data

- Leveraged C* and wide tables
- Custom Carbon Ingestion Module
- Uses Spark to query C* and perform time series math
- Custom Java Micro Service to ‘mock’ Graphite API
- Next Steps