Scalable Online Services and Data Processing Architectures

Milind Mahajan

Grab

Overview

- Online Services
  - Hosting
  - Architecture
  - Storage
  - Toolset

- Data Processing
  - Batch
  - Streaming
  - Online Machine Learning

Online Services

Hosting

- Public Cloud Providers
  - On-demand scalability
  - Pay as you go
  - Reliable and well tested platform
  - No operational overhead
  - Leading providers: Amazon AWS, Microsoft Azure, Google Cloud

- PaaS and IaaS
  - Ease of use with some limitations vs.
    Full flexibility but more work
  - Cost difference
  - Ability to switch vendors

Service Architecture Choices

- Monoliths and Microservices
  - Monolith -- single service which does everything
  - Many services -- each focused on single logical responsibility
  - Appropriate choice depends on stage
- Monoliths -- at small scale
  - Easy to understand, deploy
  - Less up-front design/architecture work
  - Infrastructure efficiency is less of an issue at small scale
  - One team
  - Local calls between components

  - Later -- everything changes with size

Microservices

- Benefits
  - Force clear interfaces and contracts
  - Allow teams to operate independently -- large orgs
  - Each service remains easy to understand
  - Allow mix of technologies
  - Decouple deployments
  - Scale each service independently
  - Improve robustness through isolation

  - Upfront design, documentation, backward compatibility
  - RPC/REST calls between services
  - Communicate only through interfaces
  - Deployment co-ordination, scale decisions -- devops

Different types of storage

- Instance based storage
- Persistent disk storage
- Object store: large blob data, cheap, standalone
- Key-value store
  - Primary key for sharding
  - Secondary key for range queries
  - High scalability, low latency
  - Limit on size of value
- Relational databases
  - Provide transactions
  - Ease of use -- powerful programming model
  - Scalability - cost

- Caching -- can be used to reduces storage access and latency

Other toolset

- Deployment
- Dashboards and visualization
- Monitoring & alerting
- Log search
- Payment gateways

Data Processing

Hadoop

- Map-Reduce programming model
- HDFS Storage
- High scale -- large data and compute
- Fault tolerance -- deals with failures
- Well-understood
- Available as hosted service from cloud providers
- Higher level platforms: Hive, Pig, Scalding

- Data is written back to HDFS between map-reduce steps
- High latency

Spark

- Fast and general cluster based data processing
- Avoids disk I/O -- keeps data in memory where possible
- Efficient data sharing
  - Much faster for iterative algorithms
  - Enables functionality such as streaming and interactive queries
- Resilient Distributed Dataset (RDD)
  - Immutable distributed collection of objects
  - Transformations form a DAG
  - Lazily evaluated after action
  - Possible to persist RDD in memory for reuse
- Fault tolerance
  - Track lineage and recompute if needed
- Powerful components: Spark SQL, MLLib, GraphX, Spark Streaming

Spark Streaming

- Break input data streams into micro-batches
- Each micro-batch is RDD
- Process each micro-batch to produce stream of results

- Abstraction is Discretized stream (DStream) of RDDs
- Stateless and stateful transforms on DStreams
- State captures information across micro-batches

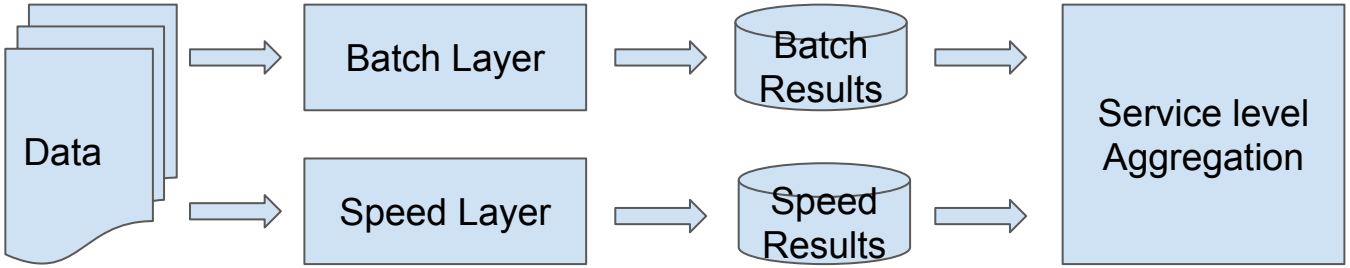
- Fault-tolerance
  - Input data is replicated
  - State is checkpointed to avoid long recompute chains
  - Trade-off between checkpoint overhead and recovery overhead

Real-time streams

- Hosted real-time streams
- Publisher pushes data: user interaction data or service generated
- Multiple consumers, low latency
- Build topologies by chaining streams
- Open source: Kafka

Lambda Architecture

- Combines the advantages of batch and stream processing



- Batch layer: high latency, high throughput, consistent
- Speed layer: low latency, (relatively) low throughput, may not be consistent
- Backfill is easier on code changes
- Results must be additive
- Data is immutable, append only

Online Machine Learning

- Online prediction: features are available only at the time of the request
- Build prediction service
- Need to ensure that features used for training and prediction match
  - Log features passed to prediction service
  - Build common libraries and configs which are used for both online and offline feature extraction
- Features need to be simple to compute in streaming
- Online training
  - Stream processing
  - Lambda architecture for adapting a batch trained model

Questions