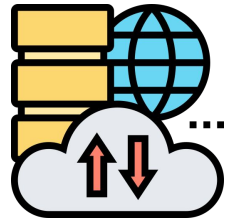
# COEN 241
# Introduction to Cloud Computing

Lecture 16 - Big Data II



# Lecture 15 Recap

- Big Data
  - What is Big Data
  - Big Data Platforms
- Online Data Processing
  - Message Queue Systems
  - Online Analytics
- Readings
  - Recommended: CCSA Chapter 8, 10.1~10.8
  - Optional: None

# What is Big Data?

- Definition: Data that contains greater **variety**, arriving in increasing **volumes** and with more **velocity**.
  - **Three Vs**
- Variety: Many types of data that are available
  - Traditional data types were structured and fit neatly in a relational database
  - Now, data comes in various unstructured formats
- Volume: Amount of data, which varies across the users
  - ~ terabytes for some, hundreds of petabytes for others
- Velocity: Fast rate at which the data arrives
  - Data often arriving in streams

# What is Big Data?

- More recently there are two new ‘V’s that emerged
- Value: Data has intrinsic value, but not useful until you analyze it
  - Need good methods and **systems** for big data analytics
- Veracity: How truthful is your data?
  - Garbage in - Garbage out

# Cloud Computing's Role in Big Data

- Cloud Computing allows Big Data to be available, scalable and fault-tolerant
- Hard to deploy & manage clusters for Big Data storage and analytics
- Cloud computing provides
  - Agility
  - Elasticity
  - Cost saving
  - Reduced complexity
  - **Platforms for big data processing**

# Big Data Platforms

- Storage (We have covered this)
  - Distributed SQL/NoSQL Databases
  - OLAP: Redshift, BigQuery
- Offline (Batch) Analytics
  - **MapReduce**
    - Hadoop
    - Spark
    - Pig, Hive
- Online (Real-Time) Analytics
  - Apache Storm
    - **Message Queue**

# Online vs Offline Analytic System

- Online: Real-time, interactive workloads where data is ingested, stored and analyzed. Addressing **Velocity** in Big Data.
- Offline: Retrospective, sophisticated analyses that may touch most or all of the data. Addressing **Volume** in Big Data.
- Both types of systems can integrate with one another
  - E.g., Online system gathers, processes and stores data for offline system
- Most companies run both types of analytics

# MapReduce Introduction

- Definition: A **programming model** and an associated implementation for processing and generating big data sets in a parallel manner
- A MapReduce program comprises of three steps:
  - Map: Filtering and sorting of data
  - Shuffle: Worker nodes redistribute data based on the output keys
    - All data with the same key goes to same worker
  - Reduce: Performing a summary operation for data with the same key
    - Counting

# MapReduce 5-Step Computation

- Prepare input: Each map processor gets assigned the input key K1 that each processor would work on, gets all the input data associated with that key.
- Map:  Map function is run exactly once for each K1 key, generating output organized by key K2
- Shuffle: Each reduce processors gets assigned the K2 key each processor should work on, gets all the Map-generated data associated with that key.
- Reduce: Reduce function is run exactly once for each K2 key.
- Produce the final output:  Collects all the Reduce output, and sorts it by K2 to produce the final outcome.

# MapReduce System (1): Hadoop

- Hadoop
  - Collection of Open-Source software for running MapReduce
    - Started in 2006
  - Hadoop Distributed File System: Used for input, output files. Block Storage
  - Hadoop YARN: Resource manager
  - Hadoop MapReduce: Implementation of MapReduce
- Pros
  - Easy to run, scale MapReduce
  - Cheap and Fast
- Cons
  - Not fast enough
  - Slow for small files

# MapReduce System (2): Spark

- Spark
  - Another Open-Source software for running MapReduce
    - Started in 2014
  - Requires a distributed storage, such as HDFS, S3 and so on.
- Pros
  - 100x faster than Hadoop for smaller workloads
  - Ideal for real-time processing
- Cons
  - Costly, requires machine with larger memory

# MapReduce Systems on the Cloud

- Amazon Elastic MapReduce
  - On-demand processing power
  - Auto-scaling
  - Easy to set up
- Google MapReduce
- Azure Serverless MapReduce via Durable Functions
  - https://docs.microsoft.com/en-us/samples/azure-samples/durablefunctions-mapreduce-dotnet/big-data-processing-serverless-mapreduce-on-azure/

# When to use MapReduce?

- MapReduce is not a one-stop solution
  - Often need to rethink your algorithm
  - Require a large cluster
  - May cost a lot more
  - May even be slower, each MapReduce step is costly
- Criteria for using MapReduce
  - **Easy to parallelize**
  - Big Data
  - If intermediate outputs are useful

# Agenda for Today

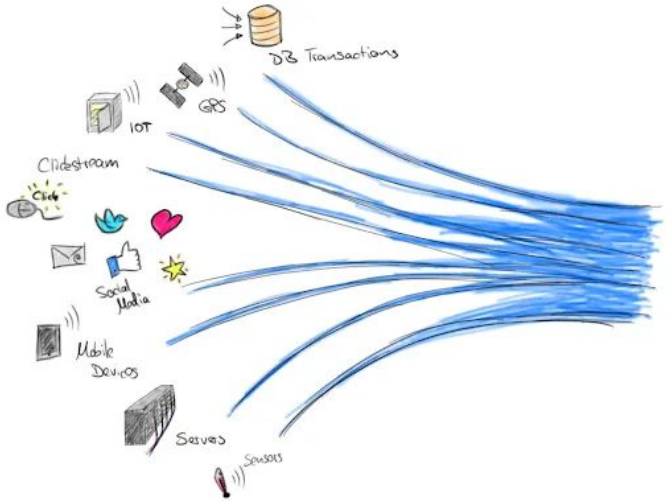
- Online Data Processing Mechanisms
- Stream Analytics
- Readings
  - Recommended: CCSA Chapter 9, 11
  - Optional
    - https://www.slideshare.net/ptgoetz/apache-storm-vs-spark-streaming
    - https://medium.com/@chandanbaranwal/spark-streaming-vs-flink-vs-storm-vs-kafka-streams-vs-samza-choose-your-stream-processing-91ea3f04675b
    - https://cloud.google.com/pubsub/docs/overview




# Online Data Processing Mechanisms

# How to Address the Velocity?

- We now have data coming in “Streams”
  - Data that is generated continuously from thousands of data sources
  - E.g., IoT devices / sensors continuously sending data




# How to Address the Velocity?

- Cannot just store all the data as it comes in
  - Lot of garbage can come in
  - Storage is costly
  - May be just too much data to handle
- Need for a system to receive the data streams, order the data and process them nicely

# How to Address the Velocity?

- There are now multiple solutions to handle the velocity issue
  - Push / Pull Ingestion Mechanisms
  - Stream Processing Frameworks
  - Data Analytics Framework

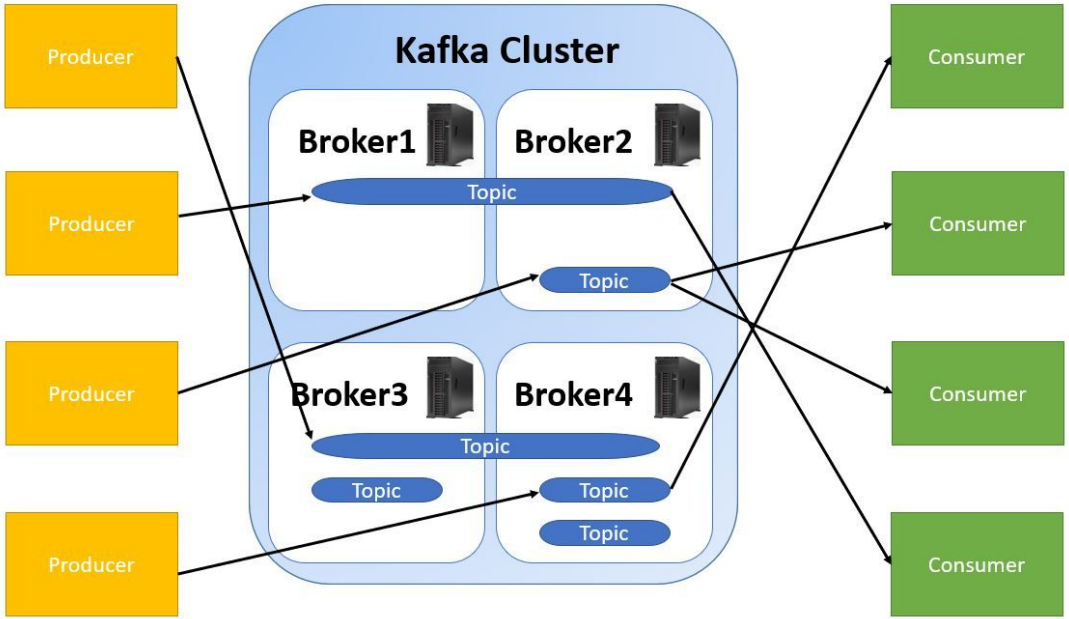
# Ingestion Mechanisms

- Data ingestion can either be a push or a pull mechanism
- Push
  - Data producers push data to a framework
  - The framework pushes data to a data sink
- Pull
  - Consumers pull the data from a messaging queue or a pub-sub framework

# Ingestion Mechanisms

- Various ***implementation*** / **frameworks** of the push & pull mechanisms
- Publish-Subscribe Framework
- Data Collection System
  - Push Online
  - Pull Offline
- Messaging Queue

# Publish-Subscribe Framework



# Publish-Subscribe Framework

- A model that comprises of **producer**, **brokers** and **consumers**
- **Producer**: Source of data and generates data to **topics**
- **Brokers**: Manages topics and handles persistence, partitioning and replication of data
- **Consumer**: Subscribes to a topic to ingest data
- **Topic**: User-defined category

# Publish-Subscribe Framework Examples

- Apache Kafka
  - Open Sourced in 2011
  - Developed at LinkedIn
  - Now a 20B company called Confluent
  - https://softwaremill.com/kafka-visualisation/
- AWS Kinesis
  - Fully-managed service on AWS
- GCP PubSub
- Azure Event Hub

# When to Use a Pub-Sub Framework

- Message Broker
- Activity Tracking
- Log Aggregation
- Stream Processing
- Real-time Event distribution
- https://cloud.google.com/pubsub/docs/overview

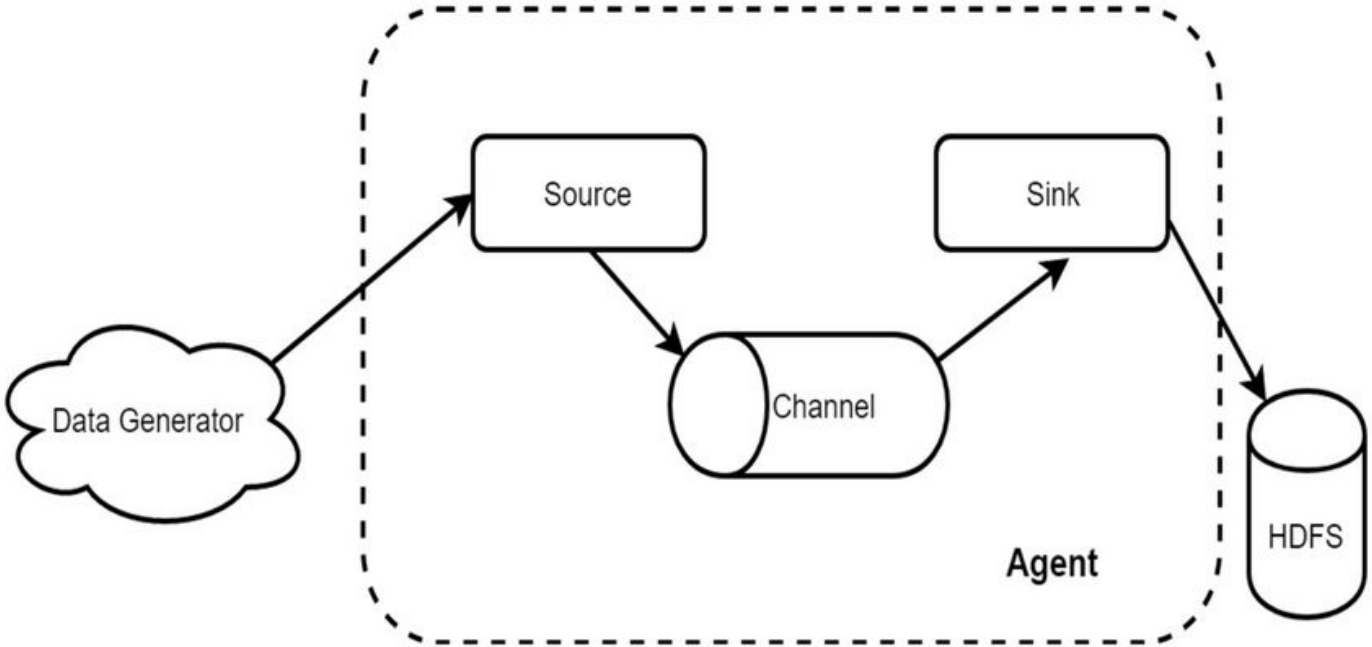
# Pros & Cons of Pub-Sub Framework

- Pros
  - **Asynchronous** in nature, no degradation of performance
  - Scalable and Flexible
  - Easy to add producers and subscribers
- Cons
  - Hard to test
  - Unexpected surge in message can cause bottlenecks
  - Require a well-defined policy for messaging and formatting

# Data Collection Framework

- Allows collecting, aggregating and moving data from various sources into a centralized data store
- Mostly a Push model implementation

# Data Collection Framework



# Data Collection Framework Examples

- Apache Flume
  - Initial Release: 2014
  - Used for multiple types of sources into HDFS sinks
    - Supports various other sources, but specialized for HDFS
- Apache Sqoop
  - Initial Release: 2009
  - Used for relational database data sources
  - Now reached end-of-life in July 2021
- Both are referred to as Hadoop ETL tools
- No similar cloud products

| Item | Sqoop | Flume |
|---|---|---|
| Basic Difference | Sqoop is a Hadoop tool that allows us to import data from relational databases into the Hadoop environment. It is primarily intended for use with RDBMS such as MySQL, Oracle, and others. | Apache Flume is a service that streams logs into a Hadoop cluster. It is primarily intended for transferring streaming data such as log files from various sources to the Hadoop environment. |
| Data Flow | Sqoop is compatible with any RDBMS that supports JDBC connectivity. Sqoop is a tool for moving data from relational databases to Hadoop in parallel. | Flume works well with streaming data sources that are generated in a Hadoop environment on a regular basis, such as log files from numerous servers. |
| Load Type | The data load in Apache Sqoop is not driven by events. | Apache Flume is an event-driven system. |
| When to Use | Sqoop is the way to go if your data is stored in Teradata, Oracle, MySQL Server, or Postgres. | When transferring large amounts of streaming data from diverse sources such as JMS or a spooling directory, Flume is a preferable option. |
| Storage | The HDFS file system is where data is imported. | Data flows from multiple channels into HDFS |
| Where to use | Sqoop is used for data imports and parallel data transfers since it copies data quickly. When we need to quickly replicate data and generate analytical results, we use Apache Sqoop. | Because of its distributed, dependable nature and highly available backup routes, Flume is used for data collection and aggregation. Flume is used to extract data from a variety of sources and evaluate patterns, as well as do sentiment analysis on server logs and social media data. |
| Architecture | The architecture of Apache Sqoop is connector-based. Sqoop Connectors understand how to connect to various data sources. | Apache Flume features agent-based architecture. An agent in Flume is responsible for fetching data. |
| Features | Sqoop supports direct input, which means it can map relational databases and import them into HBase and Hive directly. Sqoop streamlines data analysis. | Although it has a declarative configuration, it is extensible. |
| Performance | By shifting processing loads and extra storage to other systems, Apache Sqoop decreases processing loads and storage. Thus have fast performance. Sqoop parallelizes data transfer for optimal system utilization and fast performance. | Apache Flume is a highly reliable, linearly scalable, stream-oriented, fault-tolerant system with a customizable failover and recovery mechanism. Flume is a flexible data ingestion tool with a high-throughput and low-latency. |

# When to Use a Data Collection Framework

- When you want to store a large amount of for offline analytics
- When you want to store a large variety of data in a stable fashion
- When you don’t need real-time analytics or processing

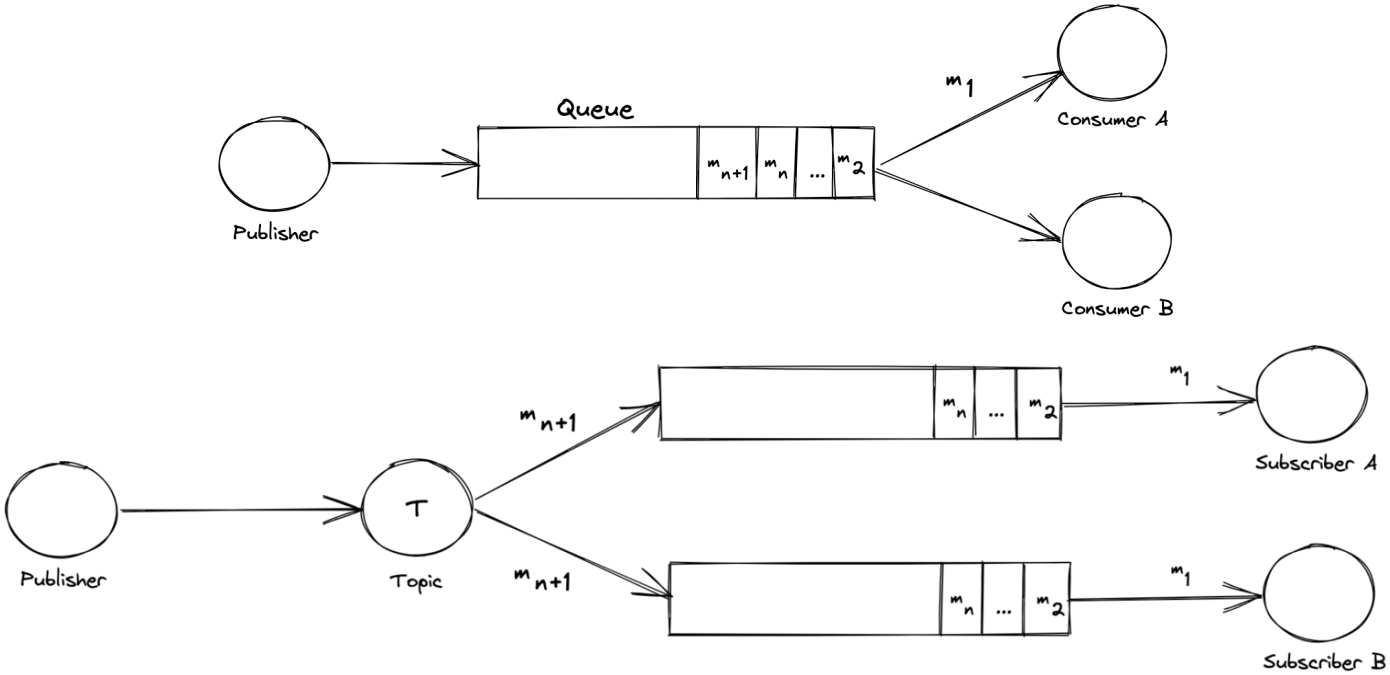
# Pros & Cons of Data Collection System

- Pros
  - Scalable
  - Steady flow of data
- Cons
  - Not really for real-time uses
  - Specialized for big data analytics, especially Hadoop

# Message Queues

- Support for Push-Pull messaging, similar to Pub-Sub
- Used to ensure that a message is **read only once by only a single consumer**
- Contrast this to Pub-Sub, where all consumers that subscribe to the topic read the message at least once

# Message Queues



# Message Queue Examples

- RabbitMQ
  - Message broker
  - Persistent message
  - https://jmcle.github.io/rabbitmq-visualizer/
  - http://tryrabbitmq.com/
- ZeroMQ
  - Socket oriented messaging system
  - Much faster than RabbitMQ
  - Not persistent
- RestMQ
  - Redis based message queue
- Amazon SQS
  - Managed queue service

# RabbitMQ Direct vs Topic Exchange

| Direct | Topic |
|---|---|
| A direct exchange delivers messages to queues based on the message routing key. | Topic exchanges route messages to one or many queues based on matching between a message routing key and the pattern that was used to bind a queue to an exchange. |
| Does not support # or * wildcard characters | Supports special characters “*” and “#” |
| Can’t behave like a topic exchange | When special characters “*” (star) and “#” (hash) aren’t used in bindings, the topic exchange will behave just like a direct one. |
| You can bind multiple queues with the same binding key.In that case, the direct exchange will behave like fanout and will broadcast the message to all the matching queues. | When a queue is bound with “#” (hash) binding key – it will receive all the messages, regardless of the routing key – like in fanout exchange. |
| It can’t do routing based on multiple criteria. | It can route based on multiple criteria |

# When to Use a Message Queue

- Simple way to decouple distributed system
  - Use message queue to send messages
  - Migrate to serverless
- Too many requests at the same time
- Event-driven programs

# Pros & Cons of Message Queues

- Pros
  - Decoupling of systems
  - Asynchronous processing
- Cons
  - Decreased availability
  - Increased complexity
  - Consistency issue
    - Message loss
    - Sequence of messages

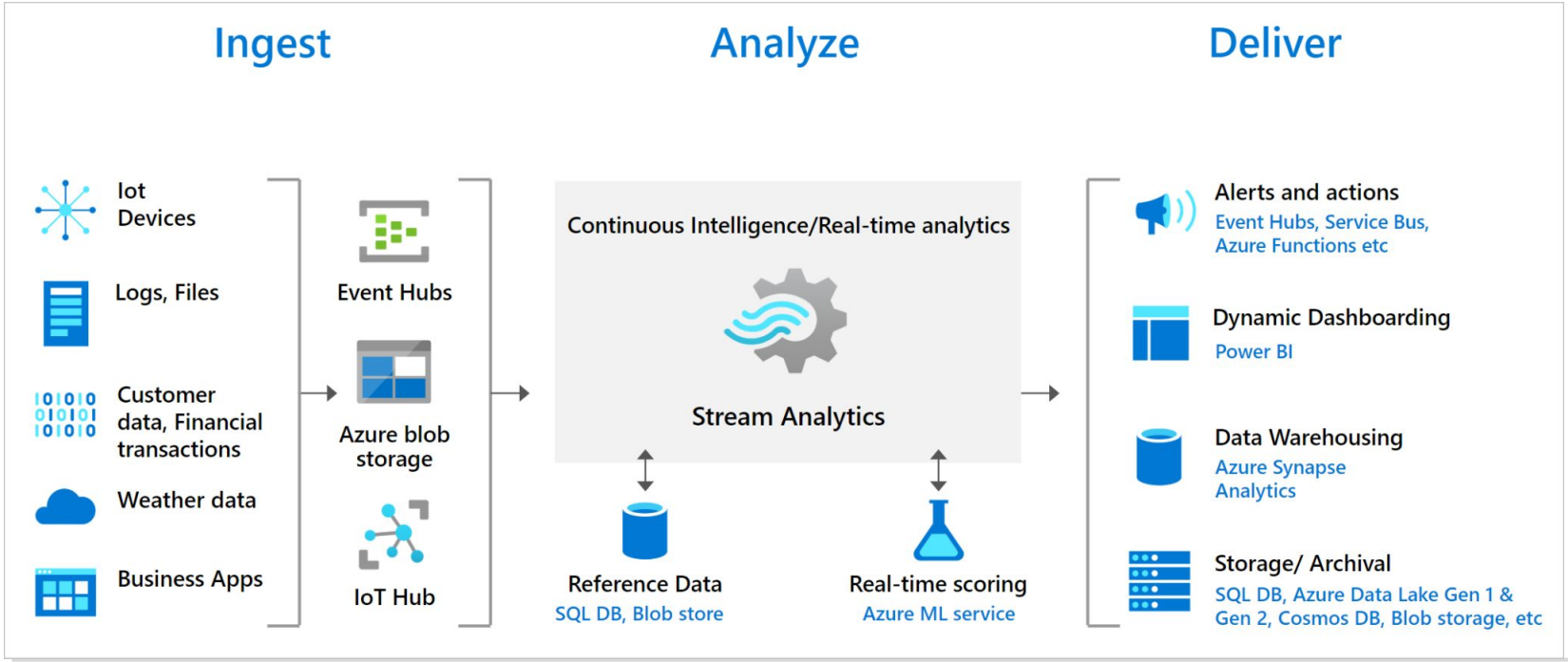



# Stream Processing Systems

# Stream Processing

- Processing and analyzing of data records continuously rather than in batches
- May include a wide variety of data sources, such as telemetry from connected devices, log files generated by customers using web applications, ecommerce transactions, or information from social networks or geospatial services
- Used for real-time aggregation and correlation, filtering, or sampling
- Also used for Machine Learning inference
  - Cyber security, fraud detection

# Stream Processing

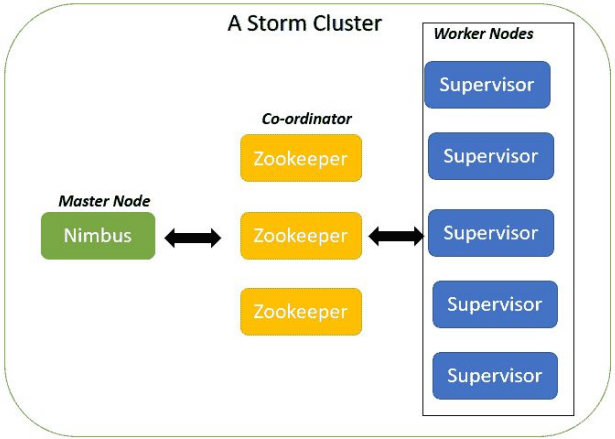


# Stream Processing Limitations

- Long-term data output rate must be just as fast, or faster, than the Long-term data input rate
  - Otherwise, the system will begin to have issues with storage and memory
- Figuring out the best way to cope with the huge amount of data that is being generated and moved
  - Reduce the number of copies
  - Optimize kernels
  - Utilize the cache hierarchy in the best way possible
  - Reducing dimensions of the machine learning model

# Stream Processing Implementations: Storm

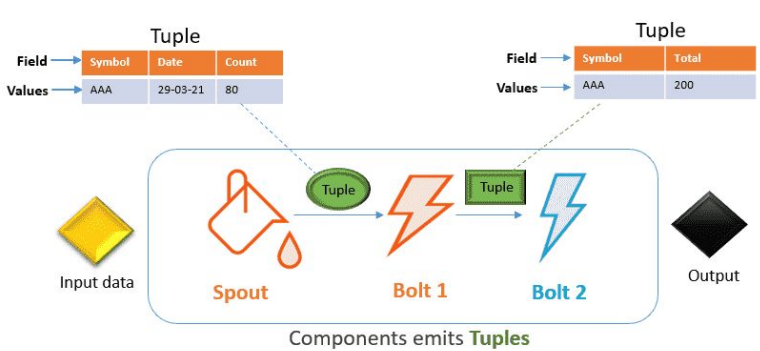
- Apache Storm
  - Open-source real-time computation system
  - Useful for:
    - Stream processing
    - Continuous computation
    - Distributed RPC
    - Real-time Analytics
- Consists of:
  - Nimbus: Master node
  - Supervisor nodes: Workers
  - Zookeeper Nodes: Consistent configuration manage
- https://storm.apache.org/Powered-By.html

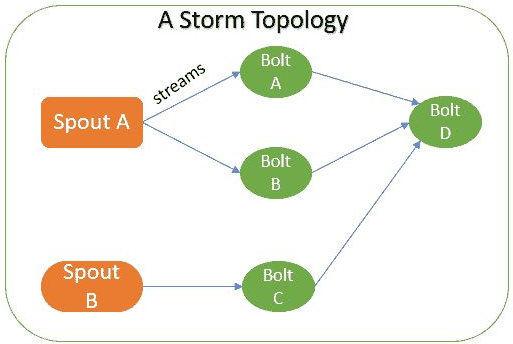



APACHE STORM™

# Stream Processing Implementations: Storm

- Apache Storm
  - Spout: Data receiver
  - Bolt: Logic
  - Tuple: Unit of data



# Stream Processing Implementations: Storm

- Pros
  - Fast
  - Fault-tolerant and reliable
- Cons
  - Stateless
  - Lower-level*

# Stream Processing Implementations: Others

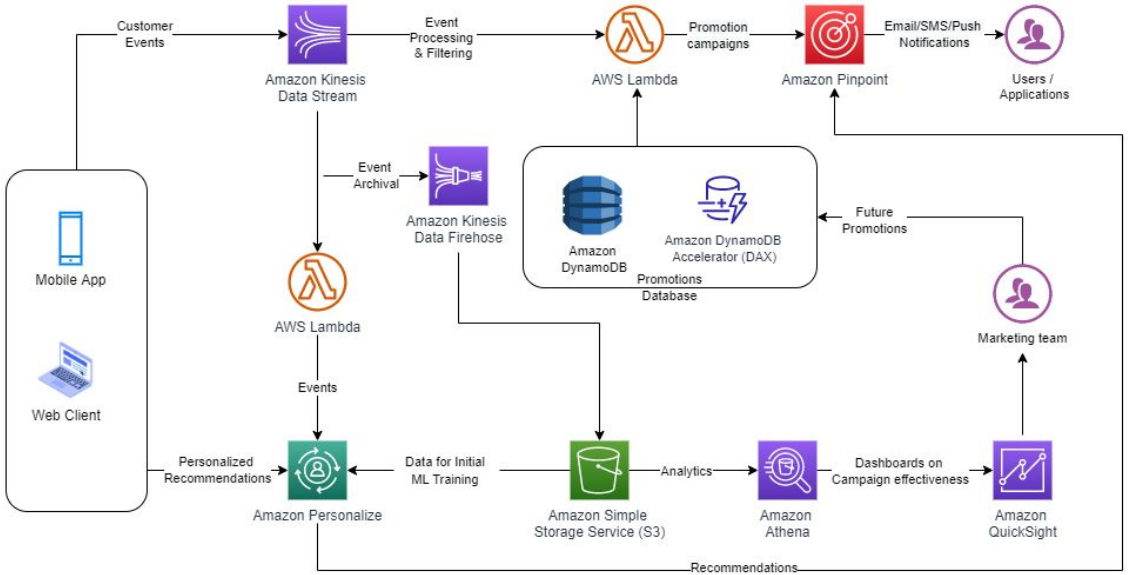
- Spark Streaming
  - Runs on top of Spark
  - Micro-batched event processing, fake streaming
  - https://www.slideshare.net/ptgoetz/apache-storm-vs-spark-streaming
- Apache Flink
  - Similar to Spark Streaming
  - Micro-batched rather than one-by-one
  - Not much adoption except Uber and Alibaba
- Apache Samza
  - Coupled with Kafka
- https://medium.com/@chandanbaranwal/spark-streaming-vs-flink-vs-storm-vs-kafka-streams-vs-samza-choose-your-stream-processing-91ea3f04675b

# Stream Processing on Cloud

- AWS Kinesis
  - AWS version of the stream processor
- Google Cloud Dataflow
- Azure Event Hub
- Can be integrated with Apache Storm

# Example Use Case

- https://aws.amazon.com/blogs/architecture/amazon-personalize-customer-outreach-on-your-ecommerce-platform/




# Agenda for Today

- Online Data Processing Mechanisms
- Stream Analytics
- Readings
  - Recommended: CCSA Chapter 9, 11
  - Optional:
    - https://www.slideshare.net/ptgoetz/apache-storm-vs-spark-streaming
    - https://medium.com/@chandanbaranwal/spark-streaming-vs-flink-vs-storm-vs-kafka-streams-vs-samza-choose-your-stream-processing-91ea3f04675b

# TODOs!

- HW 3
- Quiz 4
- Final Project

# Questions?