Architecting Big Data Solutions
Michael Carducci
@MichaelCarducci
Limitations
No Prescription
Focused on "mainstream" tools
This talk is very high-level
Ecosystem is Young
Data Science is Out of Scope
Agenda

• What is Big Data
• Big Data Design Considerations
• Big Data Solution Architecture
• Big Data Ecosystem
• Architecture Template
  • Data Acquisition
  • Persistence
  • Transform/Analytical
  • Presentation
• Sample Architectures
• Closing Thoughts
What is Big Data
What is Big Data

“90% of the data in the world today has been created in the last two years alone, at 2.5 quintillion bytes of data a day!”

–IBM Marketing Cloud, 2017
What is Big Data

- Large quantity of data
- Many sources/formats
- Unstructured
- Large Processing Load (needs to be distributed)
- Streaming/real-time processing
- Big Deployment footprint
Key Trends

- **Device Explosion**: 23.14 Billion Connected devices
- **Ubiquitous Connection**: 3.3 ZB by 2021, 278 EB/month
- **Social Networks**: 2.77 billion social media users by 2019
- **Sensor Networks**: Millions of new sensors go online every hour
- **Cheap Storage**: $0.019 Avg Cost of 1GB in 2016, 15.5 Million Times Cheaper
- **Inexpensive Computing**: $0.03 Cost/GFLOPS in 2018
Big Data

Big Opportunities
A New Way of Thinking

What, not Why
Big Data - Challenges

Volume

Variety

Velocity
Designing Big Data Solutions
Considerations
Availability
Consistency
Access Patterns
Real-time vs Historical
How Big?
Data Security
Big Data Solution Longevity

- We are still on the bleeding-edge
- Tools are still young/evolving
- Don’t “predict” the future
Future Proofing Your Education and Effort

• Don’t follow “trends” follow Value

• Look for high-level support

• Look for product/developer support

• Look for cloud options

• Look for adoption by leading companies

• Look for open APIs and Data Formats
Design Approach:
Start with the end in mind
Leverage Everything Available
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem

- **MAPREDUCE** (Processing using different languages)
- **HIVE & DRILL** (Analytical SQL-on-Hadoop)
- **MAHOUT & SPARK MLlib** (Machine learning)
- **PIG** (Scripting)
- **HBASE** (NoSQL Database)
- **KAFKA & STORM** (Streaming)
- **SOLR & LUCENE** (Searching & Indexing)
- **OOZIE** (Scheduling)
- **ZOOKEEPER & AMBARI** (Management & Coordination)

- **Resource Management**
- **Storage**

- **Flume**
  - Unstructured/Semi-Structured Data

- **Sqoop**
  - Structured Data
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Hadoop Ecosystem
Architecture Template

Transformation/Analytics

Persistence

Data Acquisition

Presentation
Data Acquisition
Data Acquisition Responsibilities

- Connect to data sources
- Data Conversion
- Filtering
- Protocol Management (handshake)
- Track Data as it moves
- Save Data to Persistence Module
- Stream Data to Processing Module
Historical Data Acquisition (Store & Forward)

• Receive from a source “at rest”
• Move data in units
• Track Completion
• Retransmit if required
Real-Time Data Acquisition (Streaming)

- Continuously moving data stream
- Throttle and Source
- Throttle for Persistence
- Inflight Storage
Data Acquisition - What to Architect

- Identification of new data
- Re-Acquisition and Retransmission
- Data Loss Tolerance
- Source Buffering
- Security in Transit
- Privacy
- Monitoring
- Throttling
- Redundancy
- Scalability
Data Acquisition - Best Practices

- Involve Source Owners
- Reliable Means for Identification of New Data
- Reliable Means for Identification of Missing Data
- APIs and Formats should be standardized as early as possible
- Consider Separate Channels for Real-Time vs Historical Data
- Pay Attention to Security and Privacy
- Use Reliable Inflight Storage
Data Acquisition - Popular Technologies

- SQOOP
- Flume
- Kafka/Storm
- Spark
- NiFi
SQOOP

- Command line tool for transferring data to/from relational data sources and hadoop
- Transfer entire database or the results of SQL
- Supports incremental data exchange
- Support for common file formats (Avro, Sequence, Parquet, CSV etc)
- Hive and HBase support
- Support Parallelism
- BLOB support
SQOOP

- Command line tool for transferring data to/from relational data sources and hadoop
- Transfer entire database or the results of SQL
- Supports incremental data exchange
- Support for common file formats (Avro, Sequence, Parquet, CSV etc)
- Hive and HBase support
- Support Parallelism
- BLOB support
Flume

• A distributed service for collecting, aggregating, and moving large amounts of log/file data

• Robust/Fault tolerant (throttling, failover, recovery etc)

• Sources can span a large number of servers

• Supports multiple sources (files, strings, http posts, etc)

• Support for multiple sinks

• Can add custom sources and sinks through code

• Supports in-flight data processing through intercepters
Flume - Benefits and Challenges

• Pros

  • Configuration Driven
  • Massively Scalable
  • Customizable through code
Flume - Benefits and Challenges

• Challenges
  • No Ordering Guarantees
  • Duplication possible
Kafka

• An open source message broker platform for real-time data feeds

• Pub/sub architecture

• Developed at LinkedIn, written in Scala

• Topics are published, multiple subscribers can be for each topic

• Ordering guarantees

• Coding required for pub/sub to interface with Kafka (not config driven)

• Replication and high Availability

• Often paired with Storm for Steam Processing
Kafka - Benefits and Challenges

• Pros

• Highly scalable realtime messaging system

• Multiple subscribers

• Ordering guarantees
Kafka - Benefits and Challenges

• Challenges

  • Coding is required for publishers and subscribers
Spark

“A fast and general engine for large-scale data processing”
Spark Components

• Spark Streaming
• Spark SQL
• MLLib
• GraphX
Spark - Benefits and Challenges

• Pros
  • Rich Ecosystem
  • Very Fast
  • Supports Streaming
  • Supports real-time Machine Learning model updates
Spark - Benefits and Challenges

• Challenges
  • In-memory processing can be expensive
  • Not “true” real-time
  • No file-management system
“A real-time integrated data logistics and simple event processing platform Apache NiFi automates the movement of data between disparate data sources and systems, making data ingestion fast, easy and secure.”
Data Acquisition - Common Sources

- RDBMS
- Logs
- Files
- Social Media
- HTTP/Rest
- Data Streams
Data Acquisition - Common Sources

• RDBMS
• Logs
• Files
• Social Media
• HTTP/Rest
• Data Streams
RDBMS - Benefits and Challenges

• Pros

  • Mature/Popular
  • Extensive Support
  • Incremental Fetches and Filtering
  • Simple transformations at Source
RDBMS - Benefits and Challenges

• Challenges

• Proprietary data platforms

• Cross-organizational boundary access limited by security
Typically Integrated with Apache Sqoop
Data Acquisition - Common Sources

- RDBMS
- Logs
- Files
- Social Media
- HTTP/Rest
- Data Streams
Files - Benefits and Challenges

• Pros

• Easy way to integrate with legacy apps

• Files easily cross organizational boundaries

• Many tools and utilities available
Files - Benefits and Challenges

• Challenges
  • Slow
  • Data Exposure
  • Potential Data/Privacy Leaks
Typically integrated with Flume
Data Acquisition - Common Sources

- RDBMS
- Logs
- Files
- Social Media
- HTTP/Rest
- Data Streams
Data Acquisition - Common Sources

- RDBMS
- Logs
- Files
- Social Media
- HTTP/Rest
- Data Streams
Data Acquisition - Common Sources

- RDBMS
- Logs
- Files
- Social Media
- HTTP/Rest
- Data Streams
REST - Benefits and Challenges

• Pros
  • de-facto standard for data exchange over the internet
  • Excellent security and scalability
  • Easy to integrate
REST - Benefits and Challenges

- Challenges
  - Not designed for real-time data
  - Stateless APIs can be “chatty”
  - Often data sources impose rate limiting
Typically integrated with Apache NiFi
Data Acquisition - Common Sources

• RDBMS

• Logs

• Files

• Social Media

• HTTP/Rest

• Data Streams
Data Acquisition - Common Sources

- RDBMS
- Logs
- Files
- Social Media
- HTTP/Rest
- Data Streams
Streaming - Benefits and Challenges

• Pros

  • Real-time, instantaneous data transfer
  • Data streams are typically deltas
  • Supported by major cloud APIs
Streaming - Benefits and Challenges

- Challenges
  - Data is typically lost if a connection is broken
  - Might need to be supplemented by historical data
  - Some providers might implement rate-limiting
Typically integrated with Kafka/Storm or Spark Streaming
Persistence
Persistence Responsibilities

- Reliable Data Storage
- Data Access
- Scalability
- Redundancy
Persistence - What to Architect

- Polyglot Storage & flexible schema
- Consistency
- Reliability
- Read-intensive vs Write Intensive
- Mutable vs immutable data
- Cataloging
- Latency
Persistence - Best Practices

• Choose platform that best supports your use case

• Keep design/schema flexible

• Keep data at lowest granularity
  • Summarize only where needed

• Consider real-time application needs

• Redundancy > Backups
Persistence

- HDFS
- HBase
- Cassandra
- S3
- Azure
- RDBMS
HDFS
HDFS
HDFS
HBase
HBase
Cassandra
S3  Simple Storage Service
Azure

Microsoft Azure
Blob Storage
RDBMS
Transform/Analytics
Transformation Responsibilities

- Build/update Machine Learning Models
- Historical processing
- Analytical querying
- Transform data into intermediate representations
- Transform data for presentation representation
Historical Data Processing

• Pull data from persistence layer
• Aggregate, summarize, and analyze data
• Save to Persistence or Presentation Layer
Real-Time Transformation/Analytics (Streaming)

- Update ML Models in real time
- Update stream analytics
Transformation - What to Architect

- Data Access latency
- Support for input volume
- Architect for future consumers of analytical data
- Architect for anticipated access patterns
- Real-time vs batch processing
Data Acquisition - Best Practices

• “What” is a question for the data scientists

• Use the best tools for the job/TIMTOOTDI
Transformation - Common Tools

- Mapreduce
- Mahout
- Spark
- Pig
- Hive/Impala
Mapreduce
Mahaout
Spark
Introducing Apache Spark

“A fast and general engine for large-scale data processing”
Spark Components

- Spark Streaming
- Spark SQL
- MLLib
- GraphX
Pig
Pig

- Abstracts Map/Reduce Complexity
- Uses PIG LATIN scripting language
- Highly extensible with UDFs
- Can perform faster than MapReduce
Oozie
Presentation
Presentation Responsibilities

- Provide analytical results in a consumable/digestible form
- Act as a reporting data source
- Bridge OLAP/OLTP Environments
- Low-latency repository
Presentation - What to Architect

- Performance
- Data Access
- Security
- Privacy
- API
- Data Refresh Frequency
- Data Lifecycle
- Availability
- Consistency
- Load
Presentation - Best Practices

- Involve Consumers
- Optimize Refresh Rate for Requirements
- Choose persistence mechanism based on access patterns and needs
- Real-time vs Digest/Summary
Presentation

• HBase
• RDBMS
• MongoDB
• Cassandra
• Redis
HBASE

- NoSQL DB built on HDFS
- Based on BigTable
- No Language - API
- Auto Sharding
HBase Data Model

- Fast access to any given row
- A row is referenced by a unique key
- Rows has a small number of **column families**
  - A column family may contain arbitrary columns
  - You can have a very large number of columns in a column family
- Each Cell can have many versions with given timestamps
- Sparse data is ok. Missing columns in a row consume no storage
Example: Web Table

Key
com.cnn.www

Contents Column Family
Contents:
- com.cnn.www
- "CNN"
- "CNN.com"

Anchor Column Family
Anchor
cnnsi.com:
- "CNN"
Anchor
my.look.ca:
- "CNN.com"
Accessing HBase

- HBase shell
- API
  - Wrappers in many language
- Spark, Hive, Pig
- Rest Service
- Thrift Service
- Avro Service
RDBMS

• Standard/mature database Technology
• Low Latency
• Good support for additional querying/filtering/transformation
• Easy to integrate
MongoDB

- NoSQL DocumentDB
- Very Fast
- Very Scalable
- Flexible Schema
- Supports Multiple Indexes
Cassandra

- NoSQL
- High Availability
- Write Scalability
- CQL Query Language
**MongoDB vs Cassandra**

<table>
<thead>
<tr>
<th>Feature</th>
<th>MongoDB</th>
<th>Cassandra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expressive object model</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Secondary indexes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>No downtime on node failure</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>High write throughput</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Redis

• Not just for caching

• Very Fast

• Distributed

• Persistent
Sample Architectures
Closing Thoughts
Thank you
Michael Carducci
@MichaelCarducci
GREAT INDIAN DEVELOPER SUMMIT
2019
Conference: April 23-26, Bangalore

Register early and get the best discounts!

www.developersummit.com
@greatindiandev
bit.ly/gidslinkedin
facebook.com/gids19
bit.ly/saltmarchyoutube
flickr.com/photos/saltmarch/