Big Data Architectural Patterns and Best Practices on AWS

Siva Raghupathy
Principal Architect and Senior Manager, Big Data Solutions Architecture
Amazon Web Services

September 2016
Agenda

Big data challenges
How to simplify big data processing
What technologies should you use?
  • Why?
  • How?
Reference architecture
Design patterns
Ever Increasing Big Data

Volume

Velocity

Variety
Big Data Evolution

Batch processing

Stream processing

Machine learning
Cloud Services Evolution

Virtual machines

Managed services

Serverless
Plethora of Tools

- Hadoop
- Hive
- Spark
- EMR
- S3
- DynamoDB
- SQS
- Cascading
- Mahout
- Presto
- Amazon Redshift
- Amazon Glacier
- RDS
- ElastiCache
- Apache HBase
- MongoDB
- R Studio
- Amazon Kinesis
- Amazon Kinesis Streams app
- Data Pipeline
- Amazon Elasticsearch Service
- Flume
- Kafka
- Kibana
- Storm
- Amazon Kinesis
- ElasticSearch
- Logstash
- ElasticSearch
- Lambda
- Amazon ML
- DynamoDB Streams
- Amazon Kinesis Analytics
- Apache Zeppelin
- Jupyter
Big Data Challenges

Is there a reference architecture?

What tools should I use?

How?

Why?
Architectural Principles

Decoupled “data bus”

• Data → Store → Process → Store → Analyze → Answers

Use the right tool for the job

• Data structure, latency, throughput, access patterns

Leverage AWS managed services

• Scalable/elastic, available, reliable, secure, no/low admin

Use Lambda architecture ideas

• Immutable (append-only) log, batch/real-time/serving layer

Big data ≠ big cost
Simplify Big Data Processing

COLLECT

STORE

PROCESS/ANALYZE

CONSUME

Time to answer (Latency)

Throughput

Cost

data

answers
COLLECT
Types of Data

Data Structures

Database Records

Search Documents

Log Files

Messages

Data Streams
What Is the Temperature of Your Data?
## Data Characteristics: Hot, Warm, Cold

<table>
<thead>
<tr>
<th></th>
<th>Hot</th>
<th>Warm</th>
<th>Cold</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Volume</strong></td>
<td>MB–GB</td>
<td>GB–TB</td>
<td>PB–EB</td>
</tr>
<tr>
<td><strong>Item size</strong></td>
<td>B–KB</td>
<td>KB–MB</td>
<td>KB–TB</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>ms</td>
<td>ms, sec</td>
<td>min, hrs</td>
</tr>
<tr>
<td><strong>Durability</strong></td>
<td>Low–high</td>
<td>High</td>
<td>Very high</td>
</tr>
<tr>
<td><strong>Request rate</strong></td>
<td>Very high</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Cost/GB</strong></td>
<td>$$-$</td>
<td>$-$–$\text{¢}$</td>
<td>$\text{¢}$</td>
</tr>
</tbody>
</table>

*Hot data* | *Warm data* | *Cold data*
Store
Types of Data Stores

- **In-memory**: Caches, Data Structure Servers
- **Database**: SQL & NoSQL Databases
- **Search**: Search Engines
- **File store**: File Systems
- **Queue**: Message Queues
- **Stream storage**: Pub/Sub Message Queues

**Applications**
- Web apps
- Mobile apps
- Data centers

**Logging**
- AWS CloudTrail
- Amazon CloudWatch

**AWS Import/Export**
- Snowball

**Messaging**
- Mobile apps
- Web apps

**Devices**
- Sensors & IoT platforms

**IoT**
- AWS IoT
Message & Stream Storage

Amazon SQS
- Managed message queue service

Apache Kafka
- High throughput distributed messaging system

Amazon Kinesis Streams
- Managed stream storage + processing

Amazon Kinesis Firehose
- Managed data delivery

Amazon DynamoDB
- Managed NoSQL database
- Tables can be stream-enabled
Why Stream Storage?

- Decouple producers & consumers
- Persistent buffer
- Collect multiple streams
- Preserve client ordering
- Parallel consumption
- Streaming MapReduce

Producer 1: 4 3 2 1 (Key = red)
Producer 2: 4 3 2 1 (Key = green)
Producer 3: 4 3 2 1 (Key = blue)
Producer n: 4 3 2 1 (Key = violet)

Counts:
- Count of red = 4
- Count of violet = 4
- Count of blue = 4
- Count of green = 4
What About Messaging?

- Decouple producers & consumers
- Persistent buffer
- Collect multiple streams
- **No** client ordering
- **No** streaming MapReduce
- **No** parallel consumption for Amazon SQS consumers
  - Amazon SNS can publish to multiple SNS subscribers (queues or AWS Lambda functions)
<table>
<thead>
<tr>
<th></th>
<th>Amazon DynamoDB Streams</th>
<th>Amazon Kinesis Streams</th>
<th>Amazon Kinesis Firehose</th>
<th>Apache Kafka</th>
<th>Amazon SQS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS managed</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Guaranteed ordering</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Delivery (deduping)</td>
<td>Exactly-once</td>
<td>At-least-once</td>
<td>At-least-once</td>
<td>At-least-once</td>
<td>At-least-once</td>
</tr>
<tr>
<td>Data retention period</td>
<td>24 hours</td>
<td>7 days</td>
<td>N/A</td>
<td>Configurable</td>
<td>14 days</td>
</tr>
<tr>
<td>Availability</td>
<td>3 AZ</td>
<td>3 AZ</td>
<td>3 AZ</td>
<td>Configurable</td>
<td>3 AZ</td>
</tr>
<tr>
<td>Scale / throughput</td>
<td>No limit / ~ table IOPS</td>
<td>No limit / ~ shards</td>
<td>No limit / automatic</td>
<td>No limit / ~ nodes</td>
<td>No limits / automatic</td>
</tr>
<tr>
<td>Parallel clients</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Stream MapReduce</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>Row/object size</td>
<td>400 KB</td>
<td>1 MB</td>
<td>Destination row/object size</td>
<td>Configurable</td>
<td>256 KB</td>
</tr>
<tr>
<td>Cost</td>
<td>Higher (table cost)</td>
<td>Low</td>
<td>Low</td>
<td>Low (+admin)</td>
<td>Low-medium</td>
</tr>
</tbody>
</table>

**Cost**

- **Hot**: Higher (table cost)
- **Low**: Low
- **Warm**: Low (table cost)
- **Cool**: Low (+admin)
Why is Amazon S3 Good for Big Data?

- Natively supported by big data frameworks (Spark, Hive, Presto, etc.)
- No need to run compute clusters for storage (unlike HDFS)
- Can run transient Hadoop clusters & Amazon EC2 Spot Instances
- Multiple & heterogeneous analysis clusters can use the same data
- Unlimited number of objects and volume of data
- Very high bandwidth – no aggregate throughput limit
- Highly available – can tolerate AZ failure
- Designed for 99.999999999% durability
- No need to pay for data replication
- Native support for versioning
- Tiered-storage (Standard, IA, Amazon Glacier) via life-cycle policies
- Secure – SSL, client/server-side encryption at rest
- Low cost
What about HDFS & Data Tiering?

• Use HDFS for very frequently accessed (hot) data
• Use Amazon S3 Standard for frequently accessed data
• Use Amazon S3 Standard – IA for less frequently accessed data
• Use Amazon Glacier for archiving cold data
In-memory/Cache, Database, Search
Anti-Pattern

Applications

RDBMS
Best Practice - Use the Right Tool for the Job

Applications

In-memory
Amazon ElastiCache
Redis
Memcached

NoSQL
Amazon DynamoDB
Cassandra
HBase
MongoDB

SQL
Amazon Aurora
Amazon RDS
MySQL
PostgreSQL
Oracle
SQL Server

Search
Amazon Elasticsearch Service
In-memory/Cache, Database, Search
What Data Store Should I Use?

Data structure → Fixed schema, JSON, key-value

Access patterns → Store data in the format you will access it

Data characteristics → Hot, warm, cold

Cost → Right cost
# Data Structure and Access Patterns

## Access Patterns

<table>
<thead>
<tr>
<th>Access Patterns</th>
<th>What to use?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put/Get (key, value)</td>
<td>In-memory, NoSQL</td>
</tr>
<tr>
<td>Simple relationships $→$ 1:N, M:N</td>
<td>NoSQL</td>
</tr>
<tr>
<td>Multi-table joins, transaction, SQL</td>
<td>SQL</td>
</tr>
<tr>
<td>Faceting, search</td>
<td>Search</td>
</tr>
</tbody>
</table>

## Data Structure

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>What to use?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed schema</td>
<td>SQL, NoSQL</td>
</tr>
<tr>
<td>Schema-free (JSON)</td>
<td>NoSQL, Search</td>
</tr>
<tr>
<td>(Key, value)</td>
<td>In-memory, NoSQL</td>
</tr>
</tbody>
</table>
## What Data Store Should I Use?

<table>
<thead>
<tr>
<th></th>
<th>Amazon ElastiCache</th>
<th>Amazon DynamoDB</th>
<th>Amazon RDS/Aurora</th>
<th>Amazon Elasticsearch</th>
<th>Amazon S3</th>
<th>Amazon Glacier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average latency</strong></td>
<td>ms</td>
<td>ms</td>
<td>ms, sec</td>
<td>ms, sec (~ size)</td>
<td>ms, sec, min (hrs)</td>
<td></td>
</tr>
<tr>
<td><strong>Typical data stored</strong></td>
<td>GB (no limit)</td>
<td>GB–TBs (64 TB max)</td>
<td>GB–TB</td>
<td>GB–TB (no limit)</td>
<td>GB–PB (no limit)</td>
<td></td>
</tr>
<tr>
<td><strong>Typical item size</strong></td>
<td>B-KB (400 KB max)</td>
<td>KB (64 KB max)</td>
<td>KB (2 GB max)</td>
<td>KB-TB (5 TB max)</td>
<td>GB (40 TB max)</td>
<td></td>
</tr>
<tr>
<td><strong>Request Rate</strong></td>
<td>High – very high (no limit)</td>
<td>Very high (no limit)</td>
<td>High</td>
<td>High</td>
<td>Low – high (no limit)</td>
<td>Very low</td>
</tr>
<tr>
<td><strong>Storage cost GB/month</strong></td>
<td>$$</td>
<td>$$</td>
<td>$$</td>
<td>$$</td>
<td>$7/10</td>
<td></td>
</tr>
<tr>
<td><strong>Durability</strong></td>
<td>Low - moderate</td>
<td>Very high</td>
<td>Very high</td>
<td>High</td>
<td>Very high</td>
<td></td>
</tr>
<tr>
<td><strong>Availability</strong></td>
<td>High 2 AZ</td>
<td>Very high 3 AZ</td>
<td>Very high 3 AZ</td>
<td>High 2 AZ</td>
<td>Very high 3 AZ</td>
<td>Very high 3 AZ</td>
</tr>
</tbody>
</table>

*Hot data* | *Warm data* | *Cold data*
Cost Conscious Design

Example: Should I use Amazon S3 or Amazon DynamoDB?

“I’m currently scoping out a project. The design calls for many small files, perhaps up to a billion during peak. The total size would be on the order of 1.5 TB per month…”

<table>
<thead>
<tr>
<th>Request rate (Writes/sec)</th>
<th>Object size (Bytes)</th>
<th>Total size (GB/month)</th>
<th>Objects per month</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>2048</td>
<td>1483</td>
<td>777,600,000</td>
</tr>
</tbody>
</table>
Cost Conscious Design

Example: Should I use Amazon S3 or Amazon DynamoDB?

https://calculator.s3.amazonaws.com/index.html
Amazon S3 or DynamoDB?

Table:

<table>
<thead>
<tr>
<th>Request rate (Writes/sec)</th>
<th>Object size (Bytes)</th>
<th>Total size (GB/month)</th>
<th>Objects per month</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>2,048</td>
<td>1,483</td>
<td>777,600,000</td>
</tr>
</tbody>
</table>

Amazon DynamoDB is a high performance non-relational database service that is easy to set up, operate, and scale. It is designed to address the core problems of database management, performance, scalability, and reliability. It also provides predictable high performance and low latency at scale.

Amazon S3 is storage for the Internet. It is designed to make web-scale computing easier for developers.

Storage:

- Storage: 1483 GB
- Reduced Redundancy Storage: 0 GB

Requests:

- PUT/COPY/POST/LIST Requests: 77760000 Requests
- GET and Other Requests: 0 Requests

Amazon S3 Service (US-East)

- Storage: $44.27
- Put/List Requests: $3888.00

Amazon DynamoDB Service (US-East)

- Provisioned Throughput Capacity: $261.69
- Indexed Data Storage: $382.61

Simple Monthly Calculator
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Request rate (Writes/sec)</th>
<th>Object size (Bytes)</th>
<th>Total size (GB/month)</th>
<th>Objects per month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>300</td>
<td>2,048</td>
<td>1,483</td>
<td>777,600,000</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>300</td>
<td>32,768</td>
<td>23,730</td>
<td>777,600,000</td>
</tr>
</tbody>
</table>

**Amazon S3 Service (US-East)**
- Storage: $44.27
- Put/List Requests: $3888.00
- Total: $3932.27

**Amazon DynamoDB Service (US-East)**
- Provisioned Throughput Capacity: $261.69
- Indexed Data Storage: $382.61
- DynamoDB Streams: $0.00
- Total: $644.30

**Amazon S3**
- Use:

**Amazon DynamoDB**
- Use:
PROCESS / ANALYZE
Analytics Types & Frameworks

Batch
- Takes minutes to hours
  - Example: Daily/weekly/monthly reports
  - Amazon EMR (MapReduce, Hive, Pig, Spark)

Interactive
- Takes seconds
  - Example: Self-service dashboards
  - Amazon Redshift, Amazon EMR (Presto, Spark)

Message
- Takes milliseconds to seconds
  - Example: Message processing
  - Amazon SQS applications on Amazon EC2

Stream
- Takes milliseconds to seconds
  - Example: Fraud alerts, 1 minute metrics
  - Amazon EMR (Spark Streaming), Amazon Kinesis Analytics, KCL, Storm, AWS Lambda

Machine Learning
- Takes milliseconds to minutes
  - Example: Fraud detection, forecast demand
  - Amazon ML, Amazon EMR (Spark ML)
<table>
<thead>
<tr>
<th></th>
<th>Amazon EMR (Spark Streaming)</th>
<th>Apache Storm</th>
<th>KCL Application</th>
<th>Amazon Kinesis Analytics</th>
<th>AWS Lambda</th>
<th>Amazon SQS Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS managed</strong></td>
<td>Yes (Amazon EMR)</td>
<td>No (Do it yourself)</td>
<td>No ( EC2 + Auto Scaling)</td>
<td>Yes</td>
<td>Yes</td>
<td>No (EC2 + Auto Scaling)</td>
</tr>
<tr>
<td><strong>Serverless</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Scale / throughput</strong></td>
<td>No limits / ~ nodes</td>
<td>No limits / ~ nodes</td>
<td>No limits / ~ nodes</td>
<td>~ KPU / automatic</td>
<td>No limits / automatic</td>
<td>No limits / ~ nodes</td>
</tr>
<tr>
<td><strong>Availability</strong></td>
<td>Single AZ</td>
<td>Configurable</td>
<td>Multi-AZ</td>
<td>Multi-AZ</td>
<td>Multi-AZ</td>
<td>Multi-AZ</td>
</tr>
<tr>
<td><strong>Programming languages</strong></td>
<td>Java, Python, Scala</td>
<td>Almost any language via Thrift</td>
<td>Java, others via MultiLangDaemon</td>
<td>ANSI SQL with extensions</td>
<td>Node.js, Java, Python</td>
<td>AWS SDK languages (Java, .NET, Python, …)</td>
</tr>
<tr>
<td><strong>Uses</strong></td>
<td>Multistage processing</td>
<td>Multistage processing</td>
<td>Single stage processing</td>
<td>Multistage processing</td>
<td>Simple event-based triggers</td>
<td>Simple event based triggers</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td>KCL and Spark checkpoints</td>
<td>Framework managed</td>
<td>Managed by KCL</td>
<td>Managed by Amazon Kinesis Analytics</td>
<td>Managed by AWS Lambda</td>
<td>Managed by SQS Visibility Timeout</td>
</tr>
</tbody>
</table>

**Fast**
## Which Analysis Tool Should I Use?

<table>
<thead>
<tr>
<th></th>
<th>Amazon Redshift</th>
<th>Amazon EMR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use case</strong></td>
<td>Optimized for Data Warehousing</td>
<td>Interactive query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>General purpose (iterative ML, RT, ..)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Batch</td>
</tr>
<tr>
<td><strong>Scale/throughput</strong></td>
<td>~Nodes</td>
<td>~ Nodes</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>Local Storage</td>
<td>Amazon S3, HDFS</td>
</tr>
<tr>
<td><strong>Optimization</strong></td>
<td>Columnar storage, Data compression, and Zone maps</td>
<td>Framework dependent</td>
</tr>
<tr>
<td><strong>Metadata</strong></td>
<td>Amazon Redshift Managed</td>
<td>Hive Meta-store</td>
</tr>
<tr>
<td><strong>BI tools supports</strong></td>
<td>Yes (JDBC/ODBC)</td>
<td>Yes (JDBC/ODBC &amp; Custom)</td>
</tr>
<tr>
<td><strong>Access controls</strong></td>
<td>Users, Groups and Access Controls</td>
<td>Integration with LDAP</td>
</tr>
<tr>
<td><strong>UDF support</strong></td>
<td>Yes (Scalar)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Access controls**

- Users, Groups and Access Controls
- Integration with LDAP

**UDF support**

- Yes (Scalar)
- Yes
What About ETL?

https://aws.amazon.com/big-data/partner-solutions/
CONSUME
Applications & API

Analysis and visualization

Notebooks

IDE
Putting It All Together
Design Patterns
Primitive: Decoupled “Data Bus”

Storage decoupled from processing

Multiple stages

Data → Store → Process → Store → Process → Answers

- process
- store
Primitive: Pub/Sub

Parallel stream consumption/processing

Data → Amazon Kinesis → Amazon Kinesis Connector Library → AWS Lambda → Apache Spark
Primitive:

Analysis framework read from or write to multiple data stores

- Data flow:
  - Amazon Kinesis
  - AWS Lambda
  - Amazon DynamoDB
  - Amazon S3

- Processing:
  - Amazon Kinesis Connector Library
  - Spark SQL
  - Spark Streaming

- Storage:
  - Amazon EMR
Real-time Analytics

Data stream → Amazon Kinesis → Amazon Kinesis Analytics → Amazon EMR → KCL App → AWS Lambda → Spark Streaming → Real-time prediction → Amazon ML → Amazon ES → Amazon RDS → Amazon DynamoDB → Amazon ElastiCache (Redis) → Fan out → Amazon SNS → Notifications

Log → Amazon S3

Alerts → App state
Interactive & Batch Analytics

- Data stream
  - Amazon Kinesis Firehose
  - Amazon S3
  - Amazon ML

Process:
- Amazon Redshift
- Amazon EMR
- Presto
- Spark

Interactive prediction
Real-time prediction

Batch prediction

Consume

- Amazon Kinesis Firehose
- Amazon S3
- Amazon EMR
- Presto
- Spark
- Hive
- Pig

Batch Analytics
Lambda Architecture

Batch & Interactive Layer

Amazon S3

Amazon Redshift

Amazon EMR

Presto

Spark

Hive

Pig

Spark

Amazon Kinesis

Amazon ML

Real-time Layer

KCL

Amazon Kinesis Analytics

AWS Lambda

Spark Streaming on Amazon EMR

Storm

Amazon ElastiCache

Amazon DynamoDB

Amazon RDS

Amazon ES

Serving Layer

Applications

Data

Amazon Kinesis

process

store
Summary

Decoupled “data bus”
- Data → Store → Process → Store → Analyze → Answers

Use the right tool for the job
- Data structure, latency, throughput, access patterns

Leverage AWS managed services
- Scalable/elastic, available, reliable, secure, no/low admin

Use Lambda architecture ideas
- Immutable (append-only) log, batch/real-time/serving layer

Big data ≠ big cost
Thank you!

aws.amazon.com/big-data