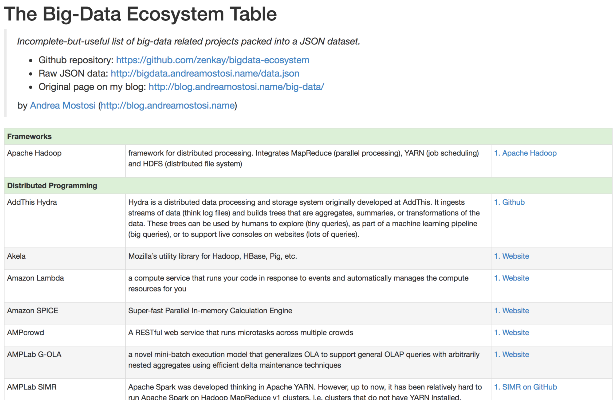
Publish-Subscribe

Principles of Functional Programming

Heather Miller

Publish-Subscribe: Why?

Recall the **Big Data Ecosystem Table**...



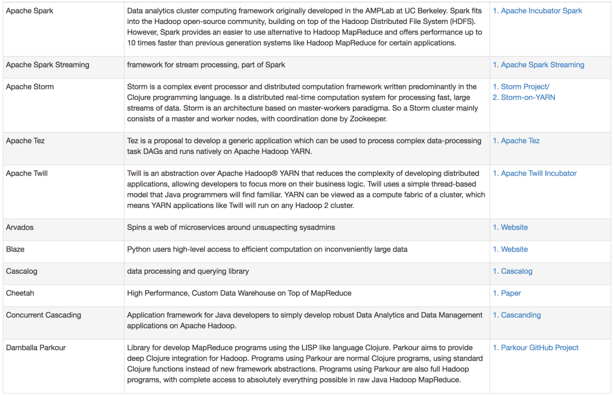
The Big-Data Ecosystem Table

> *Incomplete-but-useful list of big-data related projects packed into a JSON dataset.*
>
> - Github repository: https://github.com/zenkay/bigdata-ecosystem
> - Raw JSON data: http://bigdata.andreamostosi.name/data.json
> - Original page on my blog: http://blog.andreamostosi.name/big-data/
>
> by Andrea Mostosi (http://blog.andreamostosi.name)

Frameworks		
Apache Hadoop	framework for distributed processing. Integrates MapReduce (parallel processing), YARN (job scheduling) and HDFS (distributed file system)	1. Apache Hadoop
Distributed Programming		
AddThis Hydra	Hydra is a distributed data processing and storage system originally developed at AddThis. It ingests streams of data (think log files) and builds trees that are aggregates, summaries, or transformations of the data. These trees can be used by humans to explore (tiny queries), as part of a machine learning pipeline (big queries), or to support live consoles on websites (lots of queries).	1. Github
Akela	Mozilla's utility library for Hadoop, HBase, Pig, etc.	1. Website
Amazon Lambda	a compute service that runs your code in response to events and automatically manages the compute resources for you	1. Website
Amazon SPICE	Super-fast Parallel In-memory Calculation Engine	1. Website
AMPcrowd	A RESTful web service that runs microtasks across multiple crowds	1. Website
AMPLab G-OLA	a novel mini-batch execution model that generalizes OLA to support general OLAP queries with arbitrarily nested aggregates using efficient delta maintenance techniques	1. Website
AMPLab SIMR	Apache Spark was developed thinking in Apache YARN. However, up to now, it has been relatively hard to run Apache Spark on Hadoop MapReduce v1 clusters, i.e. clusters that do not have YARN installed.	1. SIMR on GitHub

Publish-Subscribe: Why?

Recall the **Big Data Ecosystem Table**...



Apache Spark	Data analytics cluster computing framework originally developed in the AMPLab at UC Berkeley. Spark fits into the Hadoop open-source community, building on top of the Hadoop Distributed File System (HDFS). However, Spark provides an easier to use alternative to Hadoop MapReduce and offers performance up to 10 times faster than previous generation systems like Hadoop MapReduce for certain applications.	1. Apache Incubator Spark
Apache Spark Streaming	framework for stream processing, part of Spark	1. Apache Spark Streaming
Apache Storm	Storm is a complex event processor and distributed computation framework written predominantly in the Clojure programming language. Is a distributed real-time computation system for processing fast, large streams of data. Storm is an architecture based on master-workers paradigma. So a Storm cluster mainly consists of a master and worker nodes, with coordination done by Zookeeper.	1. Storm Project/ 2. Storm-on-YARN
Apache Tez	Tez is a proposal to develop a generic application which can be used to process complex data-processing task DAGs and runs natively on Apache Hadoop YARN.	1. Apache Tez
Apache Twill	Twill is an abstraction over Apache Hadoop® YARN that reduces the complexity of developing distributed applications, allowing developers to focus more on their business logic. Twill uses a simple thread-based model that Java programmers will find familiar. YARN can be viewed as a compute fabric of a cluster, which means YARN applications like Twill will run on any Hadoop 2 cluster.	1. Apache Twill Incubator
Arvados	Spins a web of microservices around unsuspecting sysadmins	1. Website
Blaze	Python users high-level access to efficient computation on inconveniently large data	1. Website
Cascalog	data processing and querying library	1. Cascalog
Cheetah	High Performance, Custom Data Warehouse on Top of MapReduce	1. Paper
Concurrent Cascading	Application framework for Java developers to simply develop robust Data Analytics and Data Management applications on Apache Hadoop.	1. Cascanding
Damballa Parkour	Library for develop MapReduce programs using the LISP like language Clojure. Parkour aims to provide deep Clojure integration for Hadoop. Programs using Parkour are normal Clojure programs, using standard Clojure functions instead of new framework abstractions. Programs using Parkour are also full Hadoop programs, with complete access to absolutely everything possible in raw Java Hadoop MapReduce.	1. Parkour GitHub Project

Publish-Subscribe: Why?

Recall the **Big Data Ecosystem Table**...

There are many frameworks/tools/etc in this ecosystem that are often used to implement separate **microservices**.

Publish-Subscribe: Why?

Recall the **Big Data Ecosystem Table**...

There are many frameworks/tools/etc in this ecosystem that are often used to implement separate **microservices**.

However, microservices need to cooperate. For example, individual microservices typically need to:

- take tasks from some sort of list
- perform the task
- announce completion

Introducing Kafka

Can be thought of as a **distributed** publish-subscribe messaging system.

Features:

- High Availability
- High Throughput
- Scalability
- Durability (message still received, even if queue is offline)

Introducing Kafka

Can be thought of as a **distributed** publish-subscribe messaging system.

Features:

- High Availability
- High Throughput
- Scalability
- Durability (message still received, even if queue is offline)

The main value Kafka provides to data pipelines is its ability to serve as a very large, reliable buffer between various stages in the pipeline, effectively decoupling producers and consumers of data within the pipeline. This decoupling, combined with reliability, security, and efficiency, makes Kafka a good fit for most data pipelines.

Kafka Design Goals

Built at LinkedIn. Motivation:

> *"A unified platform for handling all the real-time data feeds a large company might have."*

Must haves:

- High throughput to support **high volume event feeds**.
- Support real-time processing of these feeds to create **new, derived feeds**.
- Support large data backlogs to handle periodic ingestion from **offline systems**.
- Support low-latency delivery to handle more traditional **messaging use cases**.
- Guarantee **fault-tolerance** in the presence of machine failures.

Kafka at LinkedIn (in 2014)

What type of data is being transported through Kafka?

- **Metrics**: operational telemetry data
- **Tracking**: everything a LinkedIn.com user does
- **Queuing**: between LinkedIn apps, e.g. for sending emails

Used to transport data from LinkedIn's apps to Hadoop, and back

- In total $\sim$ 200 billion events/day via Kafka
- Tens of thousands of data producers, thousands of consumers
- 7 million events/sec (write), 35 million events/sec (read) $<<<$ may include replicated events

Multiple data centers, multiple clusters. Mirroring between clusters / data centers

Ride-Sharing Application

Goal: Incentivize particular types of behavior. E.g., Get driver/rider to perform specific behavior by offering a reward.

When driver/rider accepts incentive, work is scheduled for later checking to see if task has been fulfilled. If so, reward rider/driver.

Ride-Sharing Application

Goal: Incentivize particular types of behavior. E.g., Get driver/rider to perform specific behavior by offering a reward.

When driver/rider accepts incentive, work is scheduled for later checking to see if task has been fulfilled. If so, reward rider/driver.

Clearly a queue would be useful here!

Ride-Sharing Application

Goal: Incentivize particular types of behavior. E.g., Get driver/rider to perform specific behavior by offering a reward.

When driver/rider accepts incentive, work is scheduled for later checking to see if task has been fulfilled. If so, reward rider/driver.

Clearly a queue would be useful here!

Asynchronous work scheduling:

- Web request performs some work from a mobile client
- Some work needs to be durably scheduled for some other time – logging, metrics, etc.
- Separate system reads from the queue and performs work items

Publish-Subscribe in Kafka

Basic Idea:
A model in which we provide middleware to glue requestors (**producers**) to workers (**consumers**), with much looser coupling.

Publish-Subscribe in Kafka

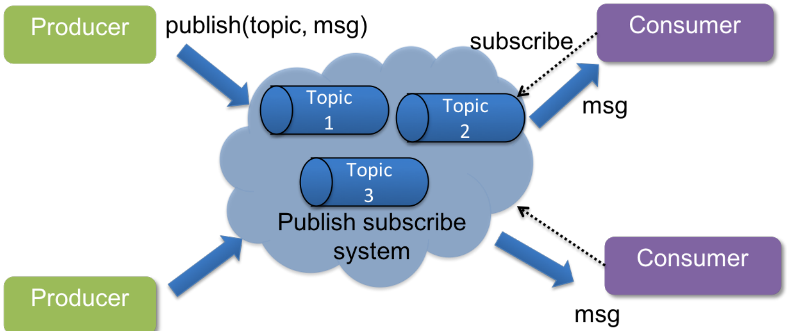
Basic Idea:
A model in which we provide middleware to glue requestors (**producers**) to workers (**consumers**), with much looser coupling.

On the Producer side:
Requests are made as **published messages**, on **topics**

On the Consumer side:
Workers monitor topics (**subscribe**) and then an idle worker can announce that it has taken on some task, and later, finished it.

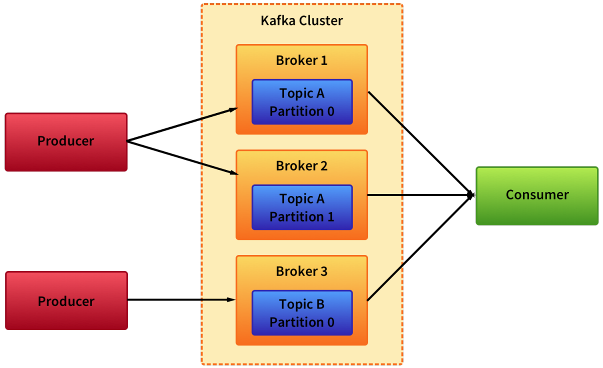
Publish-Subscribe in Kafka

Basic Idea:
A model in which we provide middleware to glue requestors to workers, with much looser coupling.



Apache Kafka Core Components

Kafka Core Components: **Producers, Consumers, Brokers,** and **Partitions**


Kafka Cluster
Broker 1
Topic A
Partition 0
Broker 2
Topic A
Partition 1
Broker 3
Topic B
Partition 0
Producer
Producer
Consumer