# Big Data Application and Framework

**Mohammad A. Hoque**
**University of Helsinki**

# Course : Big Data Frameworks

- Fundamentals of big data preprocessing and analysis
- Spark Architecture
  - File Input Format
  - Spark Lineage
  - Caching
  - Partitioning
- Rules of thumb for developing distributed ML algorithms
- Spark Streaming

# Carat

- Collaborative Analysis for Energy Hoggy or Buggy Applications in Smartphones and Recommendation.
- Collaborative analysis of Smartphone System Settings  and Recommendations
- Collaborative Analysis for Smartphone Battery Anomaly analysis and Recommendation.

# ReKnow : Data Preprocessing Pipeline

- Data Sources : Wikipedia dump, news articles, arXiv Latex Source
- Pipeline
  - Crawl → Clean Tags → HDFS (text) → Keyword Extraction Spark (Maui 2.0) → HDFS (XML)
- Issues : Maui is not thread-safe
- Working on a Spark implementation of Maui 2.0

# DAPS : Architecture

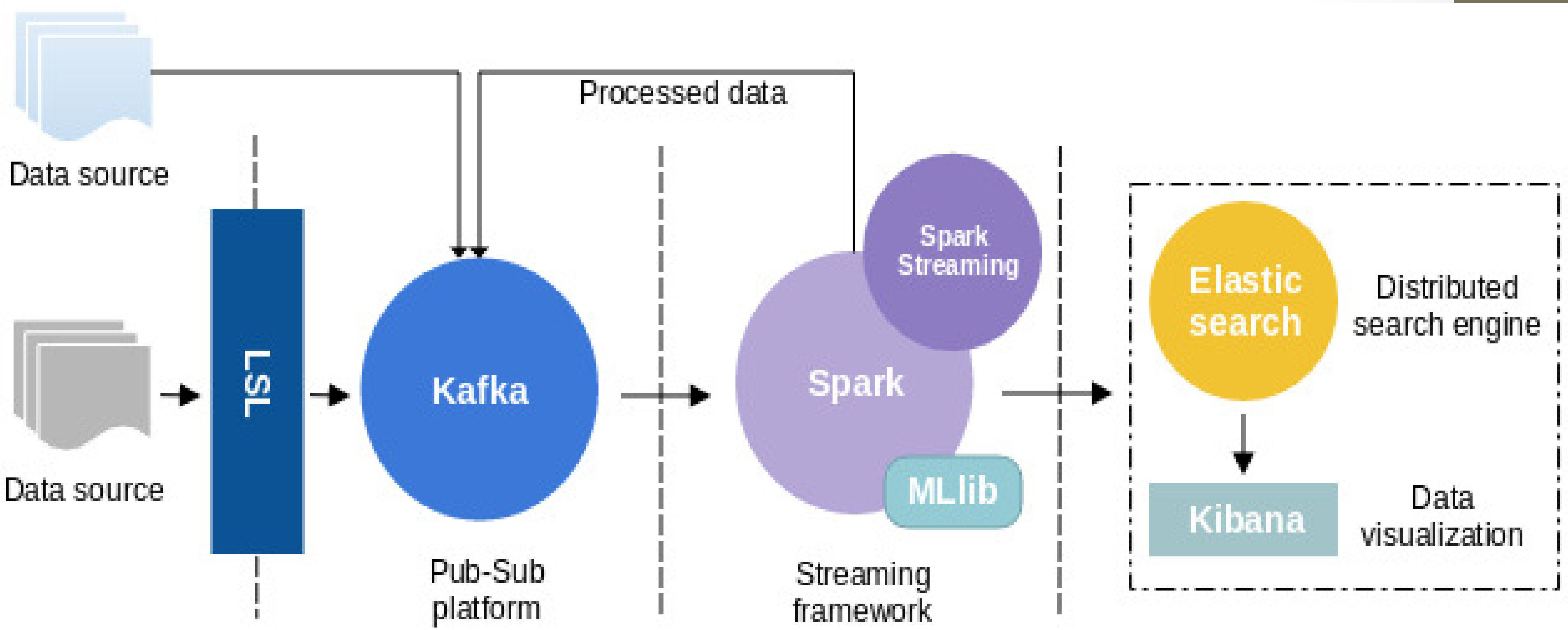


Figure Source : Maninder Pal Singh

# DAPS : Physiological Streams

- A open source framework for collecting, analyzing, and visualizing physiological time series in medical telemetry.
- Kafka : collecting streams (EEG, ECG, etc)
- Spark/ Spark Streaming : Analyzing Streams
- Elastic Search : Indexing
- Kibana : Visualization
- JSON : Data Schema to propagate data among these components.

# DAPS : API

| Operation | Meaning |
| --- | --- |
| getMovingAverage( channel, windowSize) | Return moving average of a time series for specific *channel* in a window. |
| getEuclideanDistance( channel, queryRDD) | Return Euclidean distance between two time series of particular *channel*. |
| getEuclideanDistanceForChannels( channelRDD, queryRDD) | Return Euclidean distance between two time series of multiple channels specified as *channelRDD*. |
| getDtwDistanceNaive( channelRDD, queryRDD) | Return Dynamic time warping (naive) distance between two time series as per channels specified as *channelRDD*. |
| getDtwDistanceLC( channelRDD, queryRDD, window) | Return Dynamic time warping (locality constraint) distance between two time series as per channels specified as *channelRDD*. |
| getDtwDistanceLBKeogh( channelRDD, queryRDD, radius) | Return Dynamic time warping (LB_Keogh) distance between two time series as per channels specified as *channelRDD*. |

# DAPS : API

| | |
|---|---|
| getMeanHR(ibiSeries) | Return average heart rate for Inter-beat Interval(IBI) series. |
| getMeanIBI(ibiSeries) | Return average inter-beat interval length for IBI series. |
| getRMSSD(ibiSeries) | Return root mean square by successive differences for IBI series. |
| getVariance(ibiSeries) | Return variance for IBI series. |
| getStdDev(ibiSeries) | Return standard deviation for IBI series. |
| getAllRPeaks(channel, frequency) | Return inter-beat intervals from the ECG time series. |

# CogniDA : (Spark) Rationale

- There is no way for to know whether
  - A job will finish with success or failure.
  - The amount of allocated resources is adequate for the optimal performance of an algorithm.
  - The  job will finish given the amount of resources.
  - The failure can happen due to implementation or lack of resources
  - The implementation of an algorithm is optimized or not.

# Cognida Scheduler

- Dynamic Resource Scaling
  - Extract the DAG from the scheduler and pre-execute the job.
  - Estimate, allocate the resources in each stage and create a stage-specific profile
  - Learn from the estimation and performance and create a global profile
  - Use the global resource profile for the iterative job

# CogniDa : Algorithm Type

- Some of the algorithms reduce the amount of data with iterations
  - Singular Value Decomposition

- This allows to repartition the data after each iteration.

- We can reallocate the resources after each iteration.

# Cognida : Debugging

- Per task profiling enables to find if there is any straggler.
- Per stage profiling of a job allows to understanding the performance of a job in a smaller granularity.
- This also allow to compare the performance of different implementation of the same algorithm.
- The relation between communication and computation.

# Cognida : Spark Streaming

- Processing a single time series is easy.
- Streaming frameworks are not aware of the timestamp of the samples or the speed of the stream.
- Processing multiple streams together is a challenge
  - A new stream processing system is required where batching can be dynamic.
  - Scaling of the cluster to keep up with the batching interval.

Thank You