

# **Apache Spark: A Unified Engine for Big Data Processing**

Presented by: Huanyi Chen



# Apache Spark: A **Unified Engine** for Big Data Processing

- Engine?
- Unified?

# Apache Spark: A **Unified Engine** for Big Data Processing

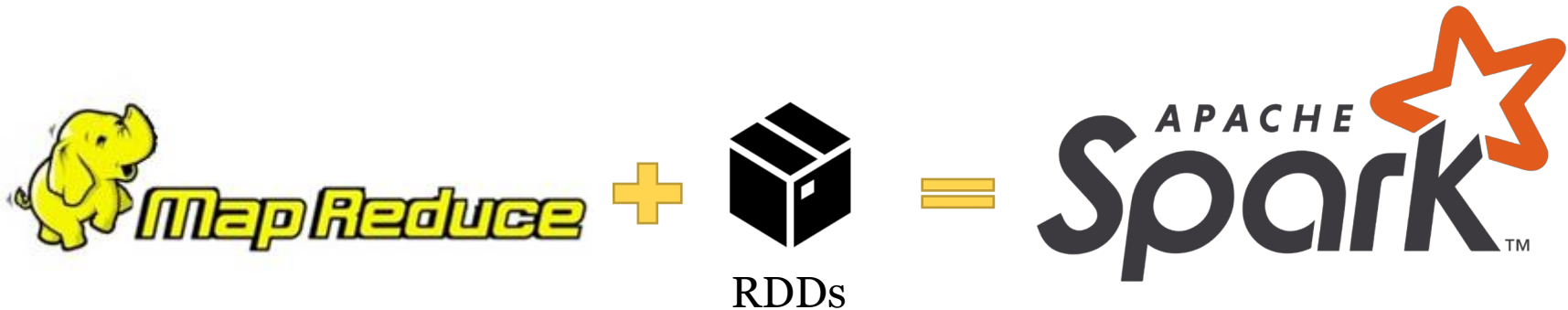
- Engine?
  - convert one form of data into other useful forms
- Unified?
  - Multiple types of conversions

# Apache Spark: A Unified Engine for Big Data Processing

- What is Apache Spark? (Engine)
- How can it make multiple types of conversions over big data? (Unified)

# What is Apache Spark?

- A framework like MapReduce
- Resilient Distributed Datasets (RDDs)



# Resilient Distributed Datasets (RDDs)

## Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma,  
Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica  
*University of California, Berkeley*

### Abstract

We present Resilient Distributed Datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude. To achieve fault tolerance efficiently, RDDs provide a restricted form of shared memory, based on coarse-grained transformations rather than fine-grained updates to shared state. However, we show that RDDs are expressive enough to capture a wide class of computations, including recent specialized programming models for iterative jobs, such as Pregel, and new applications that these models do not capture. We have implemented RDDs in a system called Spark, which we evaluate through a variety of user applications and benchmarks.

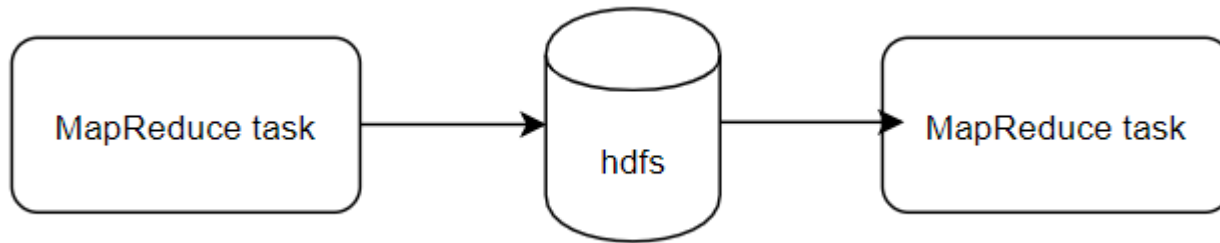
tion, which can dominate application execution times.

Recognizing this problem, researchers have developed specialized frameworks for some applications that require data reuse. For example, Pregel [22] is a system for iterative graph computations that keeps intermediate data in memory, while HaLoop [7] offers an iterative MapReduce interface. However, these frameworks only support specific computation patterns (*e.g.*, looping a series of MapReduce steps), and perform data sharing implicitly for these patterns. They do not provide abstractions for more general reuse, *e.g.*, to let a user load several datasets into memory and run ad-hoc queries across them.

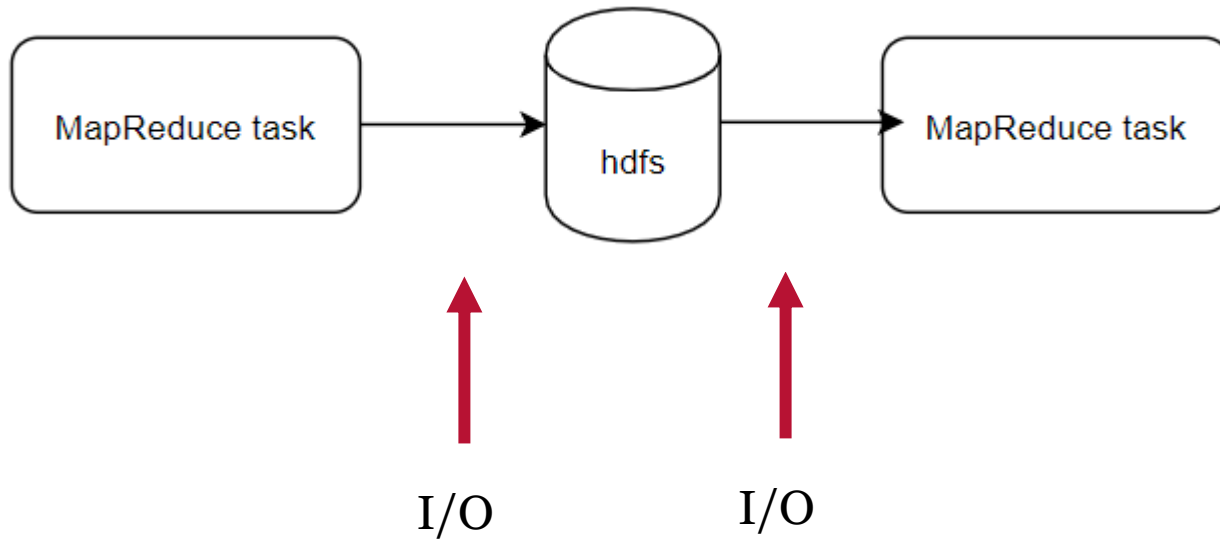
In this paper, we propose a new abstraction called *resilient distributed datasets (RDDs)* that enables efficient data reuse in a broad range of applications. RDDs are fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators.



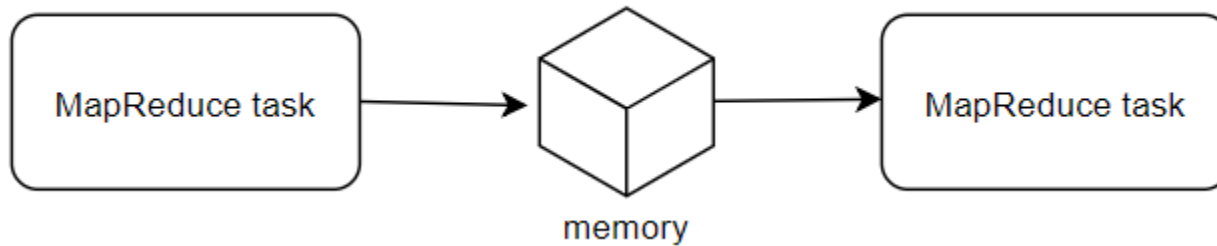
# Resilient Distributed Datasets (RDDs)



# Resilient Distributed Datasets (RDDs)



# Resilient Distributed Datasets (RDDs)



# Resilient Distributed Datasets (RDDs)

- An RDD is a read-only, partitioned collection of records
- Transformations
  - create RDDs (map, filter, join, etc.)
- Actions
  - return a value to the application
  - or export data to a storage system
- Persistence
  - Users can indicate which RDDs they will reuse and choose a storage strategy for them (e.g., in-memory storage).
- Partitioning
  - Users can ask that an RDD's elements be partitioned across machines based on a key in each record.

# Resilient Distributed Datasets (RDDs)

```
1  lines = spark.textFile("hdfs://...")
2  errors = lines.filter(_.startsWith("ERROR"))
3  errors.persist()
4
5  errors.count()
```

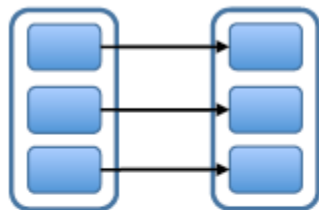
# Resilient Distributed Datasets (RDDs)

- Lineage
  - An RDD has enough information about how it was derived from other datasets.
- Narrow dependencies
  - each partition of the parent RDD is used by at most one partition of the child RDD
- Wide dependencies:
  - multiple child partitions

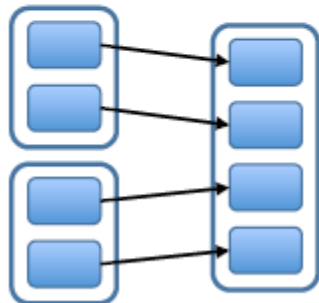


# Resilient Distributed Datasets (RDDs)

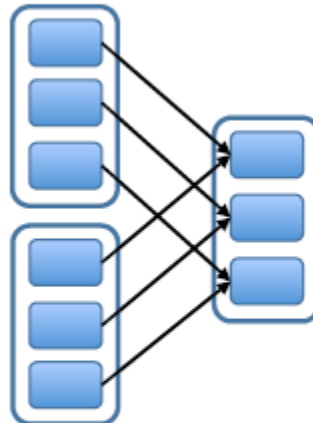
Narrow Dependencies:



map, filter

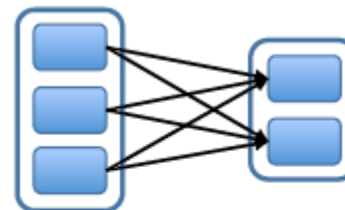


union

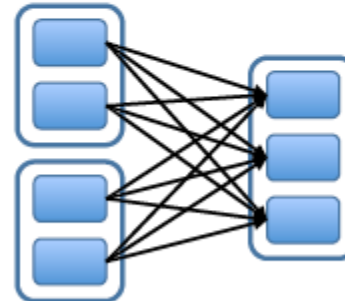


join with inputs  
co-partitioned

Wide Dependencies:

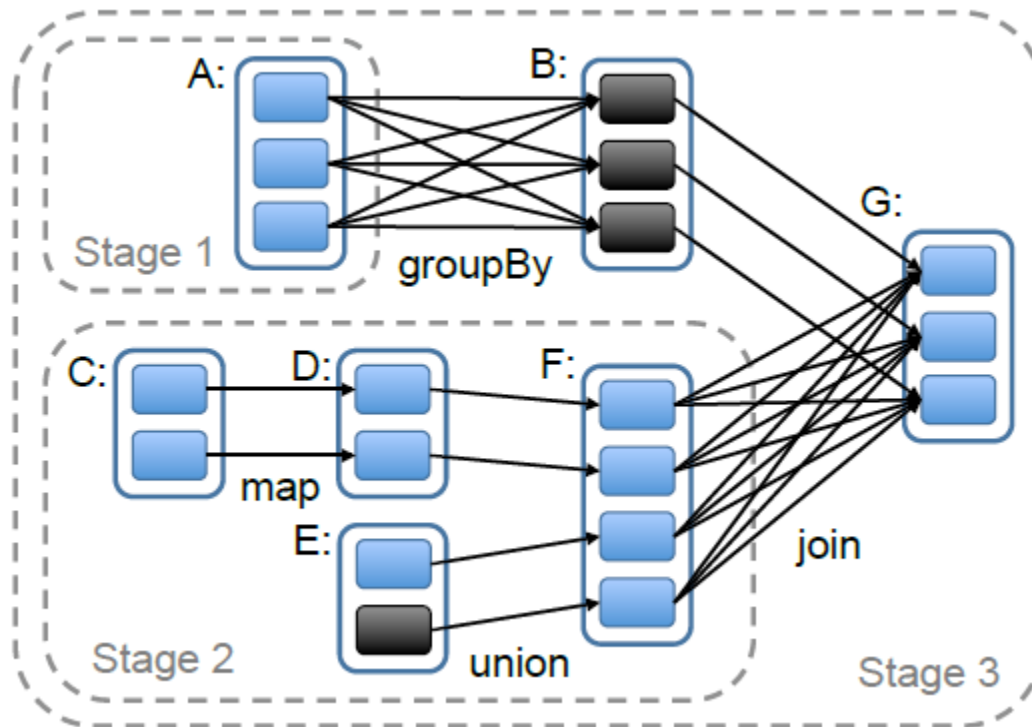


groupByKey



join with inputs not  
co-partitioned

# Resilient Distributed Datasets (RDDs)



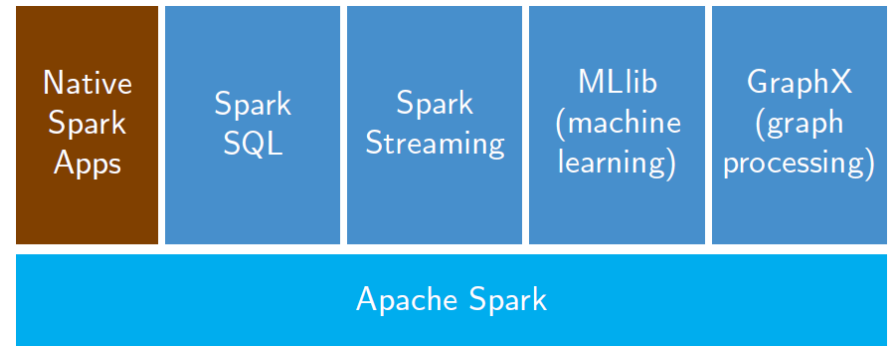
Example: run an action on RDD G

# MapReduce vs Spark

## MapReduce Ecosystem

General Batching	Specialized Systems			
	Streaming	Iterative	Ad hoc/SQL	Graph
MapReduce	Storm	Mahout	Pig	Giraph
	S4		Hive	
	Samza		Drill	
			Impala	

## Spark Ecosystem



# Higher-Level Libraries



# SQL and DataFrames

## Spark SQL: Relational Data Processing in Spark

Michael Armbrust<sup>†</sup>, Reynold S. Xin<sup>†</sup>, Cheng Lian<sup>†</sup>, Yin Huai<sup>†</sup>, Davies Liu<sup>†</sup>, Joseph K. Bradley<sup>†</sup>,  
Xiangrui Meng<sup>†</sup>, Tomer Kaftan<sup>‡</sup>, Michael J. Franklin<sup>‡</sup>, Ali Ghodsi<sup>†</sup>, Matei Zaharia<sup>†\*</sup>

<sup>†</sup>Databricks Inc.    <sup>\*</sup>MIT CSAIL    <sup>‡</sup>AMPLab, UC Berkeley

### ABSTRACT

Spark SQL is a new module in Apache Spark that integrates relational processing with Spark's functional programming API. Built on our experience with Shark, Spark SQL lets Spark programmers leverage the benefits of relational processing (*e.g.*, declarative queries and optimized storage), and lets SQL users call complex analytics libraries in Spark (*e.g.*, machine learning). Compared to previous systems, Spark SQL makes two main additions. First, it offers much tighter integration between relational and procedural processing, through a declarative DataFrame API that integrates with procedural Spark code. Second, it includes a highly extensible optimizer, Catalyst, built using features of the Scala programming language, that makes it easy to add composable rules, control code generation, and define extension points. Using Catalyst, we have built a variety of features (*e.g.*, schema inference for JSON, machine learning types, and query federation to external databases) tailored for the complex needs of modern data analysis. We see Spark SQL as an evolution of both SQL-on-Spark and of Spark itself, offering richer APIs and optimizations while keeping the benefits of the Spark programming model.

While the popularity of relational systems shows that users often prefer writing declarative queries, the relational approach is insufficient for many big data applications. First, users want to perform ETL to and from various data sources that might be semi- or unstructured, requiring custom code. Second, users want to perform advanced analytics, such as machine learning and graph processing, that are challenging to express in relational systems. In practice, we have observed that most data pipelines would ideally be expressed with a combination of both relational queries and complex procedural algorithms. Unfortunately, these two classes of systems—relational and procedural—have until now remained largely disjoint, forcing users to choose one paradigm or the other.

This paper describes our effort to combine both models in Spark SQL, a major new component in Apache Spark [39]. Spark SQL builds on our earlier SQL-on-Spark effort, called Shark. Rather than forcing users to pick between a relational or a procedural API, however, Spark SQL lets users seamlessly intermix the two.

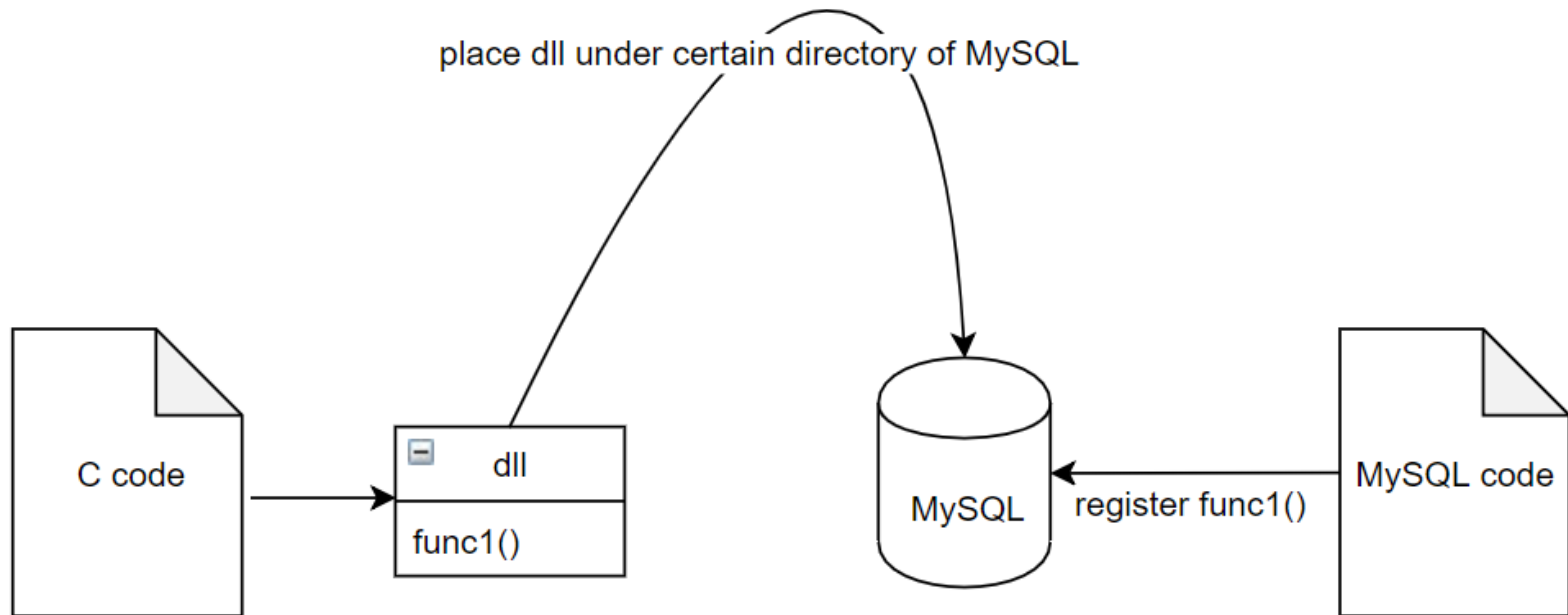
Spark SQL bridges the gap between the two models through two contributions. First, Spark SQL provides a *DataFrame API* that can perform relational operations on both external data sources and



# SQL and DataFrames

- *DataFrames = RDDs + Schema = Tables*
- Spark SQL's DataFrame API supports inline definition of user-defined functions (UDFs), without the complicated packaging and registration process found in other database systems.

# UDF in MySQL



# UDF in Spark SQL

```
val model: LogisticRegressionModel = ...  
  
ctx.udf.register("predict",  
  (x: Float, y: Float) => model.predict(Vector(x, y)))  
  
ctx.sql("SELECT predict(age, weight) FROM users")
```

# Spark Streaming

## Discretized Streams: Fault-Tolerant Streaming Computation at Scale

Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, Ion Stoica  
*University of California, Berkeley*

### Abstract

Many “big data” applications must act on data in real time. Running these applications at ever-larger scales requires parallel platforms that automatically handle faults and stragglers. Unfortunately, current distributed stream processing models provide fault recovery in an expensive manner, requiring hot replication or long recovery times, and do not handle stragglers. We propose a new processing model, *discretized streams* (D-Streams), that overcomes these challenges. D-Streams enable a *parallel recovery* mechanism that improves efficiency over traditional replication and backup schemes, and tolerates stragglers. We show that they support a rich set of operators while attaining high per-node throughput similar to single-node systems, linear scaling to 100 nodes, sub-second latency, and sub-second fault recovery. Finally, D-Streams can easily be composed with batch and interactive query models like MapReduce, enabling rich applications that combine these modes. We implement D-Streams in a system called Spark Streaming.

*faults* and *stragglers* (slow nodes). Both problems are inevitable in large clusters [12], so streaming applications must recover from them quickly. Fast recovery is even *more* important in streaming than it was in batch jobs: while a 30 second delay to recover from a fault or straggler is a nuisance in a batch setting, it can mean losing the chance to make a key decision in a streaming setting.

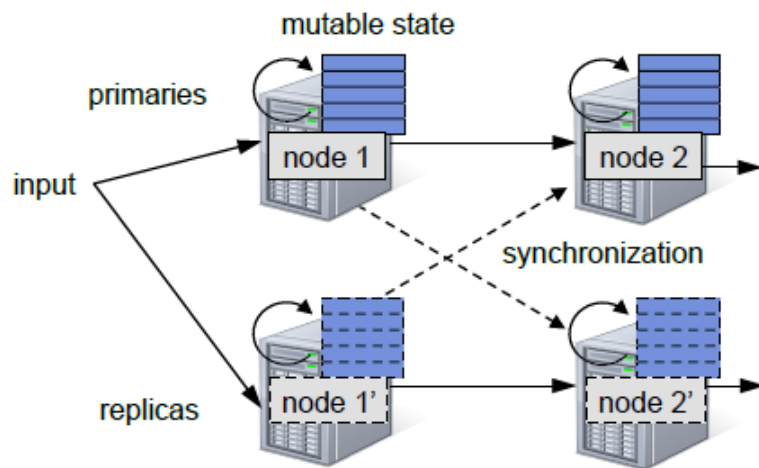
Unfortunately, existing streaming systems have limited fault and straggler tolerance. Most distributed streaming systems, including Storm [37], TimeStream [33], MapReduce Online [11], and streaming databases [5, 9, 10], are based on a *continuous operator* model, in which long-running, stateful operators receive each record, update internal state, and send new records. While this model is quite natural, it makes it difficult to handle faults and stragglers.

Specifically, given the continuous operator model, systems perform recovery through two approaches [20]: *replication*, where there are two copies of each node [5, 34], or *upstream backup*, where nodes buffer sent

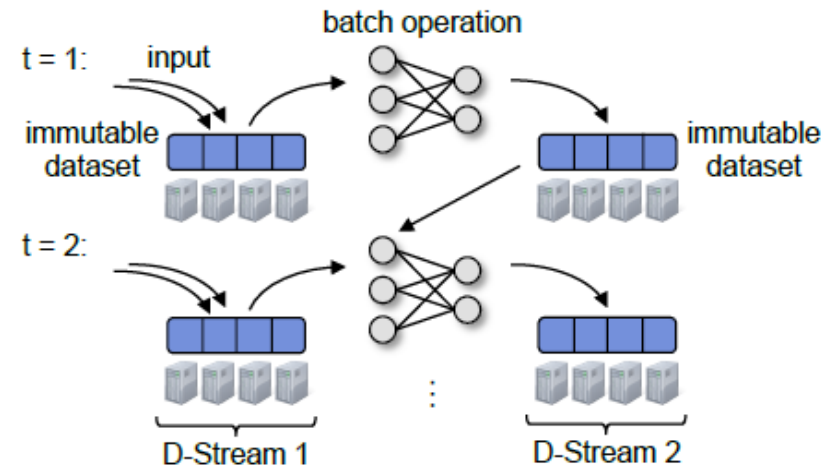


# Spark Streaming

## Continuous operator processing model



## Discretized stream processing model



# GraphX

## GraphX: Unifying Data-Parallel and Graph-Parallel Analytics

Reynold S. Xin  
Joseph E. Gonzalez

Daniel Crankshaw  
Michael J. Franklin

Ankur Dave  
Ion Stoica

UC Berkeley AMPLab  
{rxin, crankshaw, ankurd, jegonzal, franklin, istoica}@cs.berkeley.edu

### ABSTRACT

From social networks to language modeling, the growing scale and importance of graph data has driven the development of numerous new graph-parallel systems (e.g., Pregel, GraphLab). By restricting the computation that can be expressed and introducing new techniques to partition and distribute the graph, these systems can efficiently execute iterative graph algorithms orders of magnitude faster than more general data-parallel systems. However, the same restrictions that enable the performance gains also make it difficult to express many of the important stages in a typical graph-analytics pipeline: constructing the graph, modifying its structure, or expressing computation that spans multiple graphs. As a consequence, existing graph analytics pipelines compose graph-parallel and data-parallel systems using external storage systems, leading to extensive data movement and complicated programming model.

To address these challenges we introduce GraphX, a distributed graph computation framework that unifies graph-parallel and data-parallel computation. GraphX provides a small, core set of graph-parallel operators expressive enough to implement the Pregel and PowerGraph abstractions, yet simple enough to be cast in relational algebra. GraphX uses a collection of query optimization techniques such as automatic join rewrites to efficiently implement these graph-parallel operators. We evaluate GraphX on real-world graphs and workloads and demonstrate that GraphX achieves comparable performance as specialized graph computation systems, while outperforming them in end-to-end graph pipelines. Moreover, GraphX achieves a balance between expressiveness, performance, and ease of use.

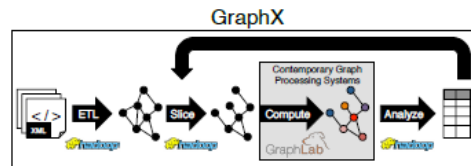


Figure 1: **Graph Analytics Pipeline:** Graph analytics is the process of going from raw data, to a graph, to the relevant subgraph, applying graph algorithms, analyzing the result, and then potentially repeating the process with a different subgraph. Currently, these pipelines compose data-parallel and graph-parallel systems through a distributed file interface. The goal of the GraphX system is to unify the data-parallel and graph-parallel views of computation into a single system and to accelerate the entire pipeline.

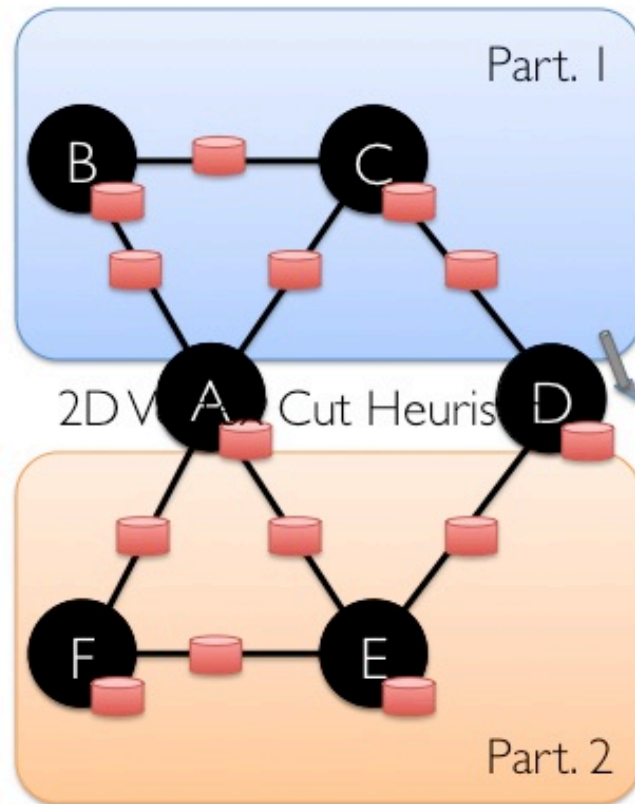
(e.g., PageRank and connected components). By leveraging the restricted abstraction in conjunction with the static graph structure, these systems are able to optimize the data layout and distribute the execution of complex iterative algorithms on graphs with tens of billions of vertices and edges.

By restricting the types of computation they express to iterative vertex-centric algorithms on a single static graph, these *graph-parallel* systems are able to achieve orders-of-magnitude performance gains over contemporary data-parallel systems such as Hadoop MapReduce. However, these same restrictions make



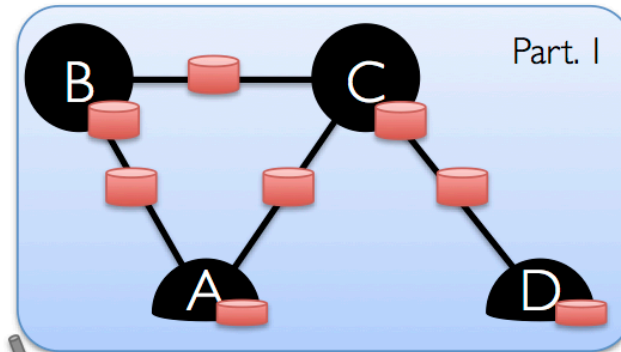
# GraphX

Property Graph

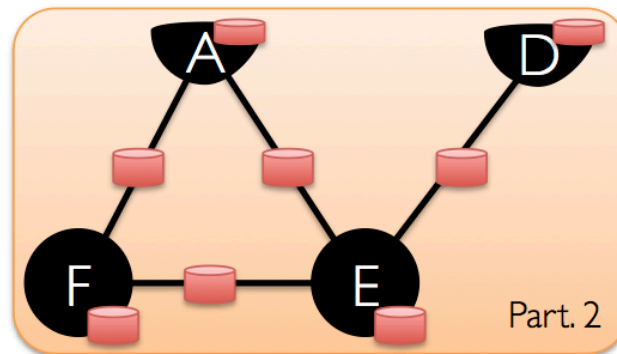


# GraphX

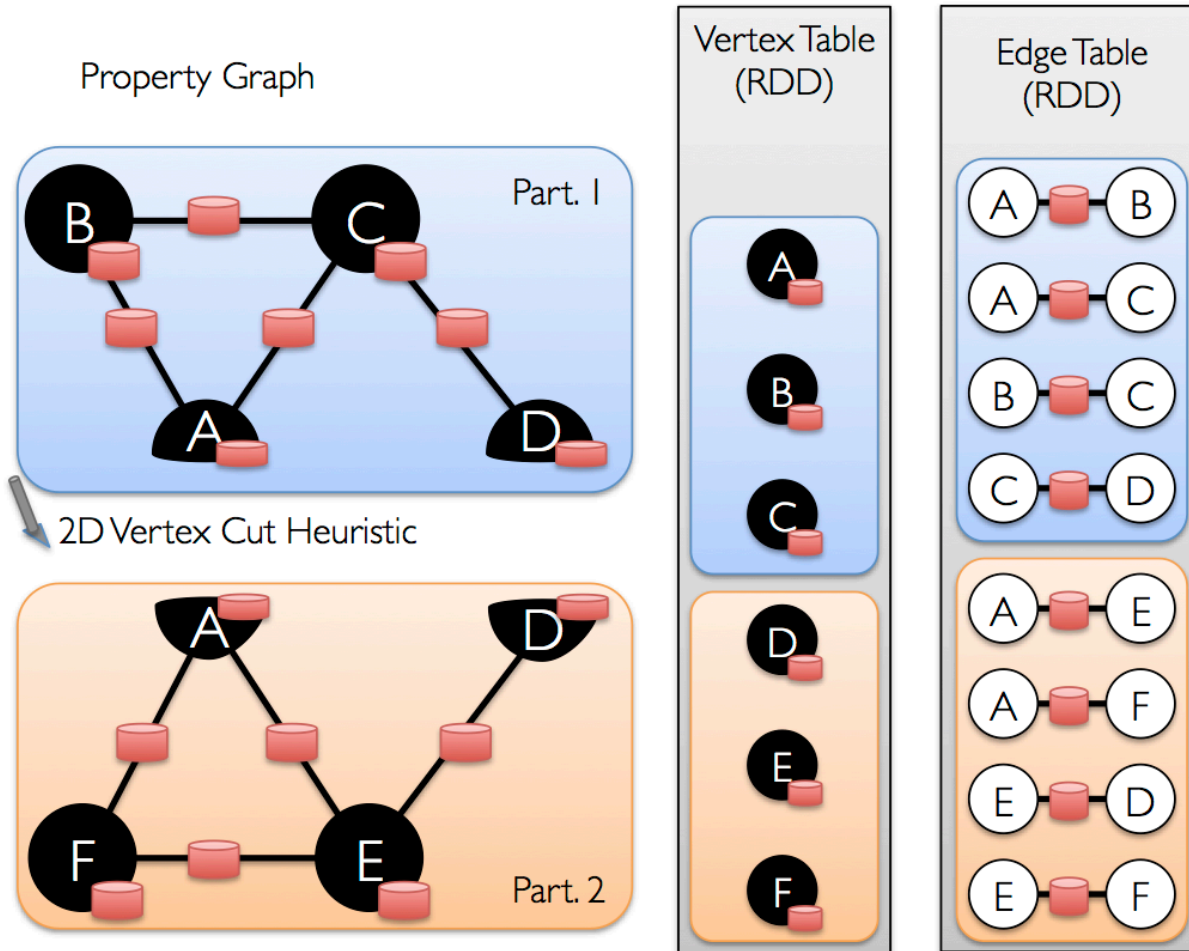
Property Graph



2D Vertex Cut Heuristic

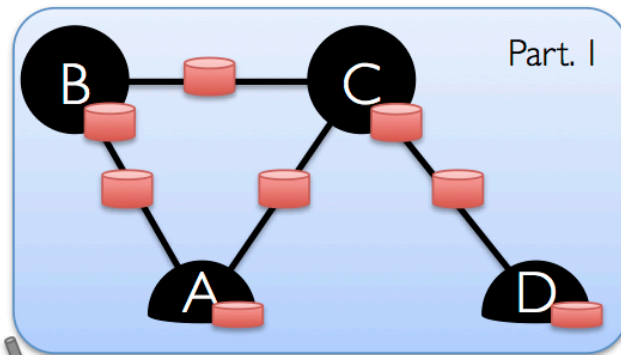


# GraphX

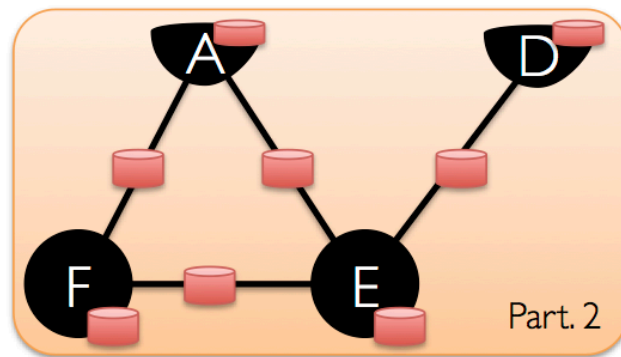


# GraphX

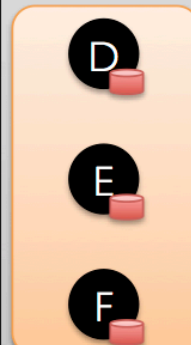
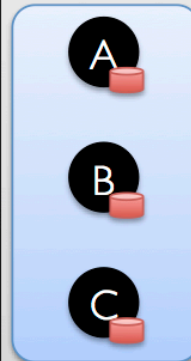
Property Graph



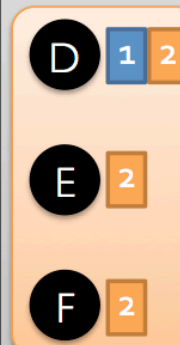
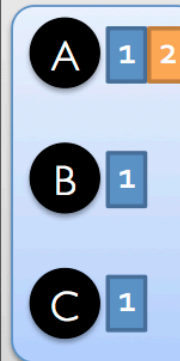
2D Vertex Cut Heuristic



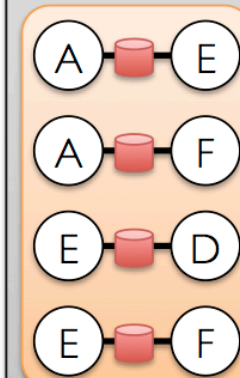
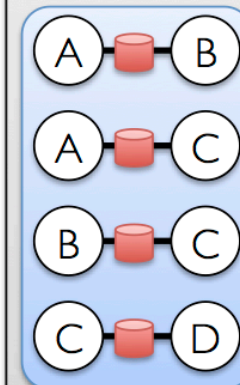
Vertex Table  
(RDD)



Routing  
Table  
(RDD)



Edge Table  
(RDD)



# GraphX

- Not able to beat specialized graph-parallel systems itself
- But outperform them in graph analytics pipeline

## MLlib: Machine Learning in Apache Spark

### Abstract

Apache Spark is a popular open-source platform for large-scale data processing that is well-suited for iterative machine learning tasks. In this paper we present MLlib, Spark's open-source distributed machine learning library. MLlib provides efficient functionality for a wide range of learning settings and includes several underlying statistical, optimization, and linear algebra primitives. Shipped with Spark, MLlib supports several languages and provides a high-level API that leverages Spark's rich ecosystem to simplify the development of end-to-end machine learning pipelines. MLlib has experienced a rapid growth due to its vibrant open-source community of over 140 contributors, and includes extensive documentation to support further growth and to let users quickly get up to speed.

**Keywords:** scalable machine learning, distributed algorithms, apache spark

# MLlib

- More than 50 common algorithms for distributed model training
- Support pipeline construction on Spark
- Integrate with other Spark libraries well



# Why use Apache Spark?

- Ecosystem
- Competitive performance
- Low cost in sharing data
- Low latency of MapReduce Steps
- Control over bottleneck resources



# Apache Spark in 2016

- Apache Spark applications range from finance to scientific data processing and combine libraries for SQL, machine learning, and graphs.
- Apache Spark has grown to 1,000 contributors and thousands of deployments from 2010 to 2016.

# Apache Spark Today

## SparkR: Scaling R Programs with Spark

Shivaram  
Xiangrui Meng

## Matrix Computations and Optimization in Apache Spark

## KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics

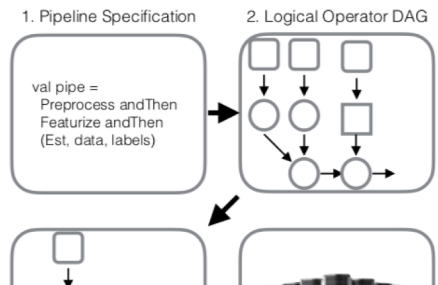
Evan R. Sparks\*, Shivaram Venkataraman\*, Tomer Kaftan\*<sup>‡</sup>, Michael J. Franklin\*<sup>†</sup>, Benjamin Recht\*  
{sparks,shivaram,tomerk11,franklin,brecht}@cs.berkeley.edu

\*AMPLab, Department of Computer Science, University of California, Berkeley

<sup>†</sup>Department of Computer Science, University of Chicago

<sup>‡</sup>Department of Computer Science, University of Washington

**Abstract**—Modern advanced analytics applications make use of machine learning techniques and contain multiple steps of domain-specific and general-purpose processing with high resource requirements. We present KeystoneML, a system that captures and optimizes the end-to-end large-scale machine learning applications for high-throughput training in a distributed environment with a high-level API. This approach offers increased ease of use and higher performance over existing systems for large scale learning. We demonstrate the effectiveness of KeystoneML in achieving high quality statistical accuracy and scalable training using real world datasets in several domains.



# Apache Spark: A Unified Engine for Big Data Processing

- What is Apache Spark
  - Apache Spark = MapReduce + RDDs
- How can it make multiple types of conversions over big data
  - Higher-level libraries enable Apache Spark to handle different types of big data workload

---

“Try Apache Spark if you are new to the big data processing world”

---

**Huanyi Chen**

# Q&A

- What issues will it cause by persisting data in memory? For example, garbage collection?
- What are Parallel Random Access Machine model and Bulk Synchronous Parallel model? Are these two models able to model any computation in distributed world?
- Will optimizing one library cause other libraries to lose performance?
- Is using memory as the storage really the next generation of storage?