What's Changing in Big Data?

Matei Zaharia



Background

Big data systems became a popular research topic nearly 10 years ago

• Large-scale, commodity clusters

What has changed since then?

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

ABSTRA We have de tem, a scala data-intens running on i high aggreg While sh

tributed file

vations of o ronment, be

departure fr has led us t ically different The file It is widely

for the gene

vice as well

large data

dreds of te

over a thou

by hundred

designed to

aspects of o

micro-benc

Categorie

D [4]: 3-D

General 7

Design, relia

Keyword

Fault tolera

*The autho

Permission to personal or cla not made or dis

bear this notice republish, to p

permission and SOSP'03, Octo

Copyright 200

{sanjay,hg

In this pa

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

1 Introduction

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a Google including our experiences in using it as the basis

To appear in OSDI 2004

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the map and reduce primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a map operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a reduce operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.

The major contributions of this work are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

Section 2 describes the basic programming model and gives several examples. Section 3 describes an implementation of the ManReduce interface tailored towards our cluster-based computing environment. Section 4 describes several refinements of the programming model that we have found useful. Section 5 has performance measurements of our implementation for a variety of tasks. Section 6 explores the use of MapReduce within







Open source processing engine and set of libraries



Cloud data processing service based on Apache Spark



Three Key Changes

1) Users: engineers → analysts

2 Hardware: I/O bottleneck → compute

3 Delivery: strong trend toward cloud



Changing Users

Initial Big Data Users

lying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. Programmers find the system easy to use: more than ten thousand distinct MapReduce programs have been implemented internally at Google over the past four years, and an average of one hundred thousand MapReduce jobs are executed on

Software engineers:

- Use Java, C++, etc to create large projects
- Build applications out of low-level operators



Expanding the User Base

Interpreting the Data: Parallel Analysis with Sawzall

Rob Pike, Sean Dorward, Robert Griesemer, Sean Quinl Google, Inc.

Abstract

Very large data sets often have a flat but regular structure and span mult machines. Examples include telephone call records, network logs, and web doc tories. These large data sets are not amenable to study using traditional database only because they can be too large to fit in a single relational database. On the oth of the analyses done on them c filtering, aggregation, extractio

We present a system for au expressed using a new procedu Both phases are distributed ove collated and saved to a file. T of the programming language. inherent in having data and cor

1 Introduction

Many data sets are too large, too relational database. One common petabytes of data-distributed acro comprise many records, organized might include a web page reposito system health records from thousa business transaction logs, network such as satellite imagery.

Quite often the analyses applied to less sophisticated than a general § satisfy a certain property, or extrac histograms of the values of certain



Facebook Data Infrastructure Team Abstract- The size of data sets being collected and analyzed in data. As a result we started exploring Hadoop as a technology the industry for business intelligence is growing rapidly, making traditional warehousing solutions prohibitively expensive. Hadoop [1] is a popular open-source map-reduce imple which is being used in companies like Yahoo, Facebook etc. to store and process extremely large data sets on commodity hardware. However, the map-reduce programming model is very low level and requires developers to write custom programs which are hard to maintain and reuse. In this paper, we present Hive, an open-source data warehousing solution built on top of Hadoop. Hive supports queries expressed in a SQL-like declarative language - *HiveOL*, which are compiled into mapreduce jobs that are executed using Hadoop. In addition, HiveQL enables users to plug in custom map-reduce scripts into queries. The language includes a type system with support for tables containing primitive types, collections like arrays and maps, and nested compositions of the same. The underlying IO libraries can be extended to query data in custom formats. Hive also includes a system catalog - Metastore - that contains schemas and statistics, which are useful in data exploration, query

Hive - A Petabyte Scale Data Warehouse Using

Hadoop

Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu

and Raghotham Murthy

700TB of data and is being used extensively for both reporting and ad-hoc analyses by more than 200 users per month. I. INTRODUCTION

optimization and query compilation. In Facebook, the Hive

ains tens of thousands of tables and stores over

Scalable analysis on large data sets has been core to the functions of a number of teams at Facebook - both on analytics. These products range from simple reporting and to also support Facebook product features. applications like Insights for the Facebook Ad Network, to concreted on Eccebook is critical Hive and

to address our scaling needs. The fact that Hadoop was already an open source project that was being used at petabyte scale and provided scalability using commodity hardware was a very compelling proposition for us. The same jobs that had taken more than a day to complete could now be completed within a few hours using Hadoon.

However, using Hadoon was not easy for end users. especially for those users who were not familiar with mapreduce. End users had to write map-reduce programs for simple tasks like getting raw counts or averages. Hadoop lacked the expressiveness of popular query languages like SOL and as a result users ended up spending hours (if not days) to write programs for even simple analysis. It was very clear to us that in order to really empower the company to analyze this data more productively, we had to improve the query capabilities of Hadoop. Bringing this data closer to users is what inspired us to build Hive in January 2007. Our vision was to bring the familiar concepts of tables, columns, partitions and a subset of SOL to the unstructured world of Hadoop, while still maintaining the extensibility and flexibility that Hadoop enjoyed. Hive was open sourced in August 2008 and since then has been used and explored by a number of Hadoop users for their data processing needs.

Right from the start, Hive was very popular with all users within Facebook. Today, we regularly run thousands of jobs engineering and non-engineering. Apart from ad hoc analysis on the Hadoop/Hive cluster with hundreds of users for a wide and business intelligence applications used by analysts across variety of applications starting from simple summarization the company, a number of Facebook products are also based jobs to business intelligence, machine learning applications

In the following sections, we provide more details about more advanced kind such as Facebook's Lexicon product [2]. Hive architecture and capabilities. Section II describes the As a result a flexible infrastructure that caters to the needs of data model, the type systems and the HiveQL. Section III these diverse applications and users and that also scales up in details how data in Hive tables is stored in the underlying a cost effective manner with the ever increasing amounts of distributed file system - HDFS(Hadoop file system). Section

CTION nber of organizations, innovation revolves ion and analysis of enormous data sets search logs, and click streams. Interas Amazon, Google, Microsoft, and Yaaples. Analysis of this data constitutes of the product improvement cycle. For ers who develop search engine ranking

Utkarsh Srivastava

such of their time analyzing search logs hese data sets dictates that it he stored ighly parallel systems, such as sharedrallel database products, e.g., Teradata, a, offer a solution by providing a simple and hiding the complexity of the physproducts however, can be prohibitively Besides, they wrench programmers erred method of analyzing data, namely cripts or code, toward writing declarawhich they often find unnatural, and

above, programmers have been flockocedural map-reduce [4] programming ce program essentially performs a grouplel over a cluster of machines. The les a map function that dictates how the and a reduce function that performs That is appealing to programmers about here are only two high-level declarative reduce) to enable parallel processing, ode, i.e., the map and reduce functions, ny programming language of choice, and bout parallelism.

map-reduce model has its own set of input, two-stage data flow is extremely asks having a different data flow, e.g., elegant workarounds have to be devised as to be written for even the most comprojection and filtering. These factors difficult to reuse and maintain, and in of the analysis task are obscured. Moreure of the map and reduce functions of the system to perform optimizations. ed a new language called Pig Latin that f both worlds: high-level declarative t of SQL, and low-level, procedural proScripting/query languages inspired by SQL, awk, etc

Used by new roles:

- Data scientists (technical domain experts, e.g. ML)
- Analysts (business)

Pig Latin: A Not-So-Foreign Language for Data Processing

Christopher Olston Beniamin Reed Yahoo! Research Yahoo! Research

nkins

earch

Challenges for Non-Engineers

API familiarity

Performance predictability & debugging Can't hide that it's large-scale

Access from small data tools E.g. Excel, Tableau Worse with more familiar APIs!



Case Study: Apache Spark

Cluster computing engine that generalizes MapReduce

Collection of APIs and libraries

- APIs in Scala, Java, Python and R
- Streaming, SQL, ML, graph, ...

1000+ deployments, max > 8000 nodes





Languages Used for Spark



Original Spark API

Functional API aimed at Java / Scala developers

Resilient Distributed Datasets (RDDs): distributed collections with functional transformations

lines = spark.textFile("hdfs://...") // RDD[String]
points = lines.map(line => parsePoint(line)) // RDD[Point]
points.filter(p => p.x > 100).count()



Challenge with Functional API

Looks high-level, but hides many semantics of computation

- Functions are arbitrary blocks of Java bytecode
- Data stored is arbitrary Java objects

Users can mix APIs in suboptimal ways



Which Operator Causes Most Tickets?

map	reduce	sample
filter	count	take
groupBy	fold	first
sort	reduceByKey	partitionBy
union	groupByKey	mapWith
join	cogroup	pipe
leftOuterJoin	Cross	save



Example Problem

pairs = data.map(word => (word, 1))

groups = pairs.groupByKey()

groups.map((k, vs) => (k, vs.sum))

 Materializes all groups as Seq[Int] objects

Then promptly aggregates them



Challenge: Data Representation

Java objects often many times larger than underlying fields

class User(name: String, friends: Array[Int])
new User("Bobby", Array(1, 2))





Structured APIs: DataFrames + Spark SQL



DataFrames and Spark SQL

Efficient library for structured data (data with a known schema)

• Two interfaces: SQL for analysts + apps, DataFrames for programmers

Optimized computation and storage, similar to RDBMS

SIGMOD 2015

Spark SQL: Relational Data Processing in Spark

Michael Armbrust[†], Reynold S. Xin[†], Cheng Lian[†], Yin Huai[†], Davies Liu[†], Joseph K. Bradley[†], Xiangrui Meng[†], Tomer Kaftan[‡], Michael J. Franklin[‡], Ali Ghodsi[†], Matei Zaharia[†]

[†]Databricks Inc. *MIT CSAIL [‡]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 Spark Spark SOL lets Spark programWhile 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 un-



Execution Steps





DataFrame API

DataFrames hold rows with a known schema and offer relational operations on them through a DSL

```
val c = new HiveContext()
val users = c.sql("select * from users")
val massUsers = users(users("state") === "MA")
massUsers.count() ExpressionAST
massUsers.groupBy("name").avg("age")
massUsers.map(row => row.getString(0).toUpper())
```

databricks^{*}

Why DataFrames?

Based on data frame conceptin R and Python

• Spark is the first to make this a declarative API

Integrates with other data science libraries



What Structured APIs Enable

- 1. Compact binary representation
 - Columnar, compressed format for caching; rows for processing
- 2. Optimization across operators (join ordering, pushdown, etc)
- 3. Runtime code generation







Performance



Aggregation benchmark (s)





DataFrames were released in March 2015, but already see high use:

62% of users in 2015 survey use DataFrames

69% of users use Spark SQL

SQL & Python are the top languages on Databricks



Other High-Level APIs

Machine Learning Pipelines Modular API based on scikit-learn DataFrame tokenizer + TF + LR + model

GraphFrames Relational + graph operations

Structured Streaming Declarative streaming API in Spark 2.0

Many high-level data science APIs can be declarative



Changing Hardware

Storage

Network

CPU



2010

Storage	50+MB/s
	(HDD)

Network 1Gbps

CPU ~3GHz











Summary

In 2005-2010, I/O was the name of the game

• Network locality, compression, in-memory caching

Now, compute efficiency matters even for data-intensive apps

• Getting harder with more diverse hardware, e.g. GPUs, FPGAs

Future: network cards ≅ DRAM bandwidth



Spark Effort: Project Tungsten

Optimize Apache Spark's CPU and memory usage, via:

- (1) Runtime code generation
- (2) Exploiting cache locality
- (3) Off-heap memory management







Runtime Code Generation

}

DataFrame Code / SQL

df.where(df("year") > 2015)

pointer arithmetic

Logical Expressions

GreaterThan(year#234, Literal(2015))

Low-level Bytecode

bool filter(Object baseObject) { int offset = baseOffset + bitSetWidthInBytes + 3*8L; int value = Platform.getInt(baseObject, offset); return value34 > 2015; JVM intrinsic JIT-ed to

databricks^{*}

Recent Additions

Whole-stage code generation

• Fuse across multiple operators

Optimized input / outputApache Parquet + built-in cache



Not Limited to Spark

Results from Nested Vector Language (NVL) project at MIT



Current systemsHand tuned code


Challenges

How to get this high performance while keeping the ease of use for non-programmers?

Can optimizations compose across libraries / systems?



Cloud Delivery

The Public Cloud is Here

Many Fortune 100 companies have multiple PB of data in S3

Amazon Web Services up to \$10B revenue

Especially attractive for big data

• 51% of respondents in 2015 Spark survey run on public cloud



Benefits

For cloud users:

- Purchase an end-to-end experience, not just bits
- Rapidly experiment with new solutions (same data & infrastructure)

For software vendors:

- Better products: end-to-end service, high visibility
- Fast iteration and uniform adoption



Challenges

Requires new way to build software that is not well understood by researchers (or traditional software companies)

- Multi-tenant: with untrusted tenants
- Highly available, yet with continuous updates
- Highly monitored for billing and security





Example Challenges

Deploying updates while keeping the service up

• And rolling back if needed!

Knowing whether the service is up

Unexpected use, especially by code calling APIs

Performance isolation of tenants at all levels

Little academic research these



Example: Databricks

End-to-end data processing platform based around Apache Spark

Access control, collaboration, auditing, production workflows

200+ customers and thousands of individual users





	INTEGRATED WORKSPACE			BITOOLS		USER APPLICATIONS	
IERPRISE SECURITY control, auditing, encryption	DASHBOARDS Reports	github	BOOKS , viz, oration	Qlik Q 🔅 🂠 tableau		PRODUCTION JOBS	
	MANAGED INFRASTRUCTURE		OPEN SOURCE	+	MANAGEMENT: Scalabil INTERFACES: BI tools &	DATABRICKS MANAGED SERVICES MANAGEMENT: Scalability, resilience, multi-tenancy INTERFACES: BI tools & RESTful APIs DATA INTEGRATION: Universal access without centralization	
E N Access				OUD ORAGE	DATA OVERALE CONTRACTOR	HOUSES	HADOOP / DATA LAKES



Lessons

Cloud development model is superior

• Two week releases, immediate feedback, visibility

State management is very hard at scale

• Per-tenant configuration, local data, VM images, etc

Careful testing strategy is crucial

• Feature flags, stress tests, 70/20/10 testing pyramid

Design to maximize dogfooding



Research Perspective

Computer systems is largely a social field: about interactions between users ⇔ machines, users ⇔ users, and machines ⇔ machines

Cloud greatly changes the way users develop and consume software

Not much research beyond using it to parallelize stuff



Example Research Problems

Composing security interfaces of different cloud providers

• E.g. Databricks access controls + Amazon IAM

Deterministic updates and rollback for complex systems

"Elastic-first" systems for price and demand variability



Conclusion

Big data systems made great strides since they first came out

- + They're used well beyond tech companies
- Not fully keeping up with new users & hardware

The cloud offers fantastic opportunities for research

- + People can try your new thing in production right away!
- Not much research fully embraces it



Thanks!

Databricks is Hiring Full-timers and interns matei@databricks.com

