Using Big Data for Analytics

Basanta Joshi (PhD)
Assistant Professor
Department of Electronics and Computer Engineering, Pulchowk Campus, Institute of Engineering (IOE), Pulchowk Campus, Lalitpur, Nepal
basanta@ioe.edu.np /www.basantajoshi.com.np

Types of Analytics

- ***Descriptive***: A set of techniques for reviewing and examining the data set(s) to understand the data and analyze business performance.
- ***Diagnostic***: A set of techniques for determine what has happened and why
- ***Predictive***: A set of techniques that analyze current and historical data to determine what is most likely to (not) happen
- ***Prescriptive***: A set of techniques for computationally developing and analyzing alternatives that can become courses of action – either tactical or strategic – that may discover the unexpected
- ***Decisive***: A set of techniques for visualizing information and recommending courses of action to facilitate human decision-making when presented with a set of alternatives.

	Passive			Active
Deductive	Descriptive			Diagnostic
Inductive	Predictive			Prescriptive

Copyright (except where referenced) 2014-2016
Stephen H. Kaisler, Frank Armour, Alberto Espinosa and William H. Money

Descriptive Analytics

- **Process**:
  - Identify the attributes, then assess/evaluate the attributes
  - Estimate the magnitude to correlate the relative contribution of each attribute to the final solution
  - Accumulate more instances of data from the data sources
  - If possible, perform the steps of evaluation, classification and categorization quickly
  - Yield a measure of adaptability within the OODA loop
- At some threshold, crossover into diagnostic and predictive analytics

http://v1shal.com/content/25-cartoons-give-current-big-data-hype-perspective/

"Data don't make any sense,
we will have to resort to statistics."

Copyright (except where referenced) 2014-2016
Stephen H. Kaisler, Frank Armour, Alberto Espinosa and William H. Money

Descriptive Analytics

Descriptive Analytics

Descriptive analytics

What is happening

Descriptive Analytics

Diagnostic Analytics

- Process:
  - Begin with descriptive analytics
  - Extract patterns from large data quantities via data mining
  - Correlate data types for explanation of near-term behavior – past and present
  - Estimate linear/non-linear behavior not easily identifiable through other approaches.
- Example: by classifying past insurance claims, estimate the number of future claims to flag for investigation with a high probability of being fraudulent.

Copyright (except where referenced) 2014-2016
Stephen H. Kaisler, Frank Armour, Alberto Espinosa and William H. Money

Predictive Analytics

- Process:
  - Begin with descriptive AND diagnostic analytics
  - Choose the right data based on domain knowledge and relationships among variables
  - Choose the right techniques to yield insight into possible outcomes
  - Determine the likelihood of possible outcomes given initial boundary conditions
  - Remember! Data driven analytics is non-linear; do NOT treat like an engineering project

Predictive Analytics

Source: SAS

Predictive Analytics Process Cycle

Common Predictive Analytics Methods

- Regression:
  - Predicting output variable using its cause-effect relationship with input variables. OLS Regression, GLM, Random forests, ANN etc.

- Classification:
  - Predicting the item class. Decision Tree, Logistic Regression, ANN, SVM, Naïve Bayes classifier etc.

- Time Series Forecasting:
  - Predicting future time events given past history. AR, MA, ARIMA, Triple Exponential Smoothing, Holt-Winters etc.

Common Predictive Analytics Methods

- Association rule mining:

– Mining items occurring together. Apriori Algorithm.

- Clustering:

– Finding natural groups or clusters in the data. K-means, Hierarchical, Spectral, Density based EM algorithm Clustering etc.

- Text mining:

– Model and structure the information content of textual sources. Sentiment Analysis, NLP

Prescriptive Analytics

- **Process**:
  - Begin w/ predictive analytics
  - Determine what should occur and how to make it so
  - Determine the mitigating factors that lead to desirable/undesirable outcomes
  - "What-if" analysis w/ local or global optimization
  - Ex: Find the best set of prices and advertising frequency to maximize revenue
  - Ex: And, the right set of business moves to make to achieve that goal

"Make it so"

Copyright (except where referenced) 2014-2016
Stephen H. Kaisler, Frank Armour, Alberto Espinosa and William H. Money

Analytics Status

Prescriptive Analytics

- **Blend of descriptive and predictive analytics**

Prescriptive Analytics

- scientific disciplines that comprise Prescriptive Analytics

Advanced Analytics

Copyright (except where referenced) 2014-2016
Stephen H. Kaisler, Frank Armour, Alberto Espinosa and William H. Money

Crowdsourcing: A New Analytic?

- Tasks normally performed by employees are outsourced via an *open call* to a large, <u>self-selected</u> community
- Some examples
  - Netflix prize
  - InnoCentive: solve R&D challenges
  - DARPA Network Challenge
- Follows an AI blackboard model
- Distributed co-creation has become a mainstream technology
- <u>Issue</u>: What metrics do we use to assess the results?
- <u>Issue</u>: How robust are the results?
- And, more issues to be addressed ….

Copyright (except where referenced) 2014-2016
Stephen H. Kaisler, Frank Armour, Alberto Espinosa and William H. Money

Crowdsourcing: Wikipedia: An Example

Ref: Mary Meeker. KPCB, presentation at All Things D.

Copyright (except where referenced) 2014-2016
Stephen H. Kaisler, Frank Armour, Alberto Espinosa and William H. Money

BDA-18

Moving Analytics To the Edge

- Traditional analysis: data is stored and then analyzed
  - Usually at some central location or a few distributed locations
  - The cost and time to move large amounts of data may render it obsolete or of little worth
- Moving analytics to the frontier of the domain:
  - (Near) real-time analysis and decisions are required
  - Streaming massive amounts of data is expensive, fraught with error: microsecond latency, millions of events
  - We just can't store it all
  - Perishability is a key factor
  - May only really need the synthesized, aggregate information, not the raw data
  - True data-driven analysis
- Example: Pushing analytics into cameras for images, full motion video analysis, motion correction, 3D perception, …

Copyright (except where referenced) 2014-2016
Stephen H. Kaisler, Frank Armour, Alberto Espinosa and William H. Money

Edge Device Analytics

- Filtering and compression processes are often tied to the downstream analytical requirements
  - Means the filtering and compression algorithms must be dynamic
- How close to the edge can we push the filtering and compression algorithms?
  - What (additional) data do these algorithms need to be effective?
  - How do we measure the efficiency of these algorithms?
  - Does the in situ hardware have the computational capacity to support such algorithms?
  - How much data correction can we do at the edges?
- Challenges:
  - How can we summarize streaming data?
  - How fast can we determine changes in the incoming data?
  - How fast can we adapt to changes in the data stream?
  - How fast can we affect the environment based on what we see?

NOTE: Not the same as IBM Edge Analytics!!

Streaming Analytics

- Streaming data: analytics must (often) occur in real time, as the data passes through the sensing/collecting device:
  - Allows you to identify and examine patterns of interest as the data is being created.
  - Can yield instant insight and immediate (re)action.
- **"Real-time is for robots"**
  - Joe Hellerstein, Professor of Computer Science at UC Berkeley.
  - "If you have people in the loop, it's **not** real time. Most people take a **second or two** to react, and that's plenty of time for a traditional transactional system to handle input and output."

Location Analytics

- What is it?
  - Augmenting mission-critical, enterprise business systems with complementary content, mapping, and geographic capabilities
  - Mapping & Visualization: use maps as the media to visualize data
  - Spatial analytics: merging GIS w/ other types of analytics
  - Find spatio-temporal patterns indicative of physical activities or social behavior
  - Data/information enrichment: add maps, imagery, demographics, consumer and lifestyle data, environment and weather, social media, etc.
- Ubiquity of GPS on cellphones, cars, wristwatches, laptops, tablets, etc.

Ref: http://www.esri.com/software/location-analytics

Ref: Kerr & Nelson,ESRI International User Conference, July 2012

Location Analytics: After a Regime Change …

- Immediate transition from "normal operations in an urban environment" to "government in place"
  - Need Civil Affairs support for managing city/region relief efforts
- No existing analysis, planning, and management tool to assist civil Government and Relief Officials
  - A lack of coherent view will lead to failure and, possibly, continuing deterioration of government, law and order, civil systems, etc…
- Why?
  - (nearly) complete destruction of governing regime
  - Disintegration of social/governmental institutions: water, education, health, law enforcement, financial, food
  - Society reorganized overnight to adapt to the new power structure, which may/may not include former government personnel
    - Ex: Sadr handed out food; created immediate constituency and had people talking about him
- Cannot win "hearts and minds" if you cannot "feed and house them"

Web Analytics

- What is it?
  - Now: The study of the behavior of web users
  - Future: The study of one mechanism for how society makes decisions
  - Example: Behavior of Web Users
    - How many people clicked on Ebola (or related terms in the past 2 months)
    - Their location, their dwell time, the number of sites they examined, the difficulty or complexity of the material on the web site
    - What can this tell us about popular concern about Ebola?
    - Can it help decision makers to better present information and decisions
  - Commercially, it is the collection and analysis of data from a web site to determine which aspects of the website achieve the business objectives

Web Analytics

- Challenge: When people start getting more of their information from the Web than from radio/tv/papers:
  - How do we design web pages to influence the opinions of people?
    - The web is not a neutral medium
  - How do we measure the influence of a web page's contents on user opinions?
    - Does # visitors translate into a viable influence metric?
    - Does dwell time translate into influence?
    - Are opinions more or less influenced as the visitor clicks through a sequence of pages?
    - How many different web sites (each of one or more pages) does a user searching a particular topic click through?
  - How do we measure the difficulty and/or complexity of the material presented on a web page(s)?
    - What is the corollary to the Flesch-Kincaid Score for web pages?
  - How do we design "good" web pages to increase their Google Page Rank score?

Visual Analytics

- *Visual analytics*: the science of analytical reasoning facilitated by interactive visual interfaces

Where are the walkers going?

Remember: The eye can be easily fooled and, thereby, fo ol the brain.

Visual Analytics

- *Visual analytics*: an evolving discipline which is driving new ways of presenting data and information to the user.
- *Visual analytics*: "the science of analytical reasoning facilitated by interactive visual interfaces" (Thomas and Cook 2005)
- Visual analytics:
  - the formation of visual metaphors in combination with a human information discourse (interaction)
  - that enables detection of the expected and discovery of the unexpected within massive, dynamically changing information spaces. (Wong and Thomas 2004)
- Visual analytics provides the "last 12 inches" between the masses of information and the human mind that enables us to make decisions.

Visual Analytics

- Why is it hard?
  - You can only see 2D because your screen is 2D
- To visualize k-dimensional data:
  - Divide the screen into multiple 2D regions and show pair-wise correlations across selected dimensions
  - Project k-dimensional data into 2D
- Projection Methods (Dimension Reduction):
  - PCA, MDS, LDA, LLE, many others …
- Many others! Usually, try to preserve distances in 2D as they exist in k-D

Visualization Analytics:

Copyright (except where referenced) 2014-2016
Stephen H. Kaisler, Frank Armour, Alberto Espinosa and William H. Money

Basic approaches to querying and exploring data using higher level tools built on top of a Hadoop Platform

Adopted from slides by By Perry Hoekstra, Jiaheng Lu, Avinash Lakshman, Prashant Malik, and Jimmy Lin

Relational
Non-relational
Analytic
Mapr
Piccolo
Hadoop
Dryad
Brisk
Hadapt
Infobright
Netezza
ParAccel
SAP Sybase IQ
Teradata
EMC
Calpont
IBM InfoSphere
Aster Data
Greenplum
VectorWise
HP Vertica
Operational
Oracle
IMB DB2
SQL Server
JustOne
InterSystems
Progress
Objectivity
Versant
Document
Lotus Notes
MarkLogic
McObject
MySQL
Ingres
PostgreSQL
SAP Sybase ASE
EnterpriseDB
NoSQL
CouchDB
MongoDB
RavenDB
Cloudant
Key Value
Couchbase
Riak
Redis
Membrain
Cassandra
Voldemort
BerkeleyDB
'as-a-Service'
App Engine Datastore
SimpleDB
Big tables
Hypertable
HBase
Graph
InfiniteGraph
Neo4J
GraphDB
NewSQL
HandlerSocket
Akiban
Amazon RDS
MySQL Cluster
SQL Azure
Clustrix
Drizzle
Database.com
GenieDB
Xeround
FathomDB
ScalArc
Schooner MySQL
CodeFutures
Tokutek
ScaleBase
NimbusDB
Continuent
VoltDB
Translattice
Data Cache
SPRAIN
Cloud Enable-ment
Data Grid/Cache
Terracotta
GigaSpaces
Oracle Coherence
memcached
IBM eXtreme Scale
GridGain
ScaleOut
Vmware GemFire
InfiniSpan
CloudTran

History of the World, Part 1

- Relational Databases – mainstay of business
- Web-based applications caused spikes
  - Especially true for public-facing e-Commerce sites
- Developers begin to front RDBMS with memcache or integrate other caching mechanisms within the application (ie. Ehcache)

Scaling Up

- Issues with scaling up when the dataset is just too big
- RDBMS were not designed to be distributed
- Began to look at multi-node database solutions
- Known as ‘scaling out’ or ‘horizontal scaling’
- Different approaches include:
  - Master-slave
  - Sharding

Scaling RDBMS – Master/Slave

- Master-Slave
  - All writes are written to the master. All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large data sets can pose problems as master needs to duplicate data to slaves

Scaling RDBMS - Sharding

- Partition or sharding
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards

Other ways to scale RDBMS

- Multi-Master replication
- INSERT only, not UPDATES/DELETES
- No JOINs, thereby reducing query time
  - This involves de-normalizing data
- In-memory databases

What is NoSQL?

- Stands for **N**ot **O**nly **SQL**
- Class of non-relational data storage systems
- Usually do not require a fixed table schema nor do they use the concept of joins
- All NoSQL offerings relax one or more of the ACID properties (will talk about the CAP theorem)

Why NoSQL?

- For data storage, an RDBMS cannot be the be-all/end-all
- Just as there are different programming languages, need to have other data storage tools in the toolbox
- A NoSQL solution is more acceptable to a client now than even a year ago
  - Think about proposing a Ruby/Rails or Groovy/Grails solution now versus a couple of years ago

How did we get here?

- Explosion of social media sites (Facebook, Twitter) with large data needs
- Rise of cloud-based solutions such as Amazon S3 (simple storage solution)
- Just as moving to dynamically-typed languages (Ruby/Groovy), a shift to dynamically-typed data with frequent schema changes
- Open-source community

Dynamo and BigTable

- Three major papers were the seeds of the NoSQL movement
  - BigTable (Google)
  - Dynamo (Amazon)
    - Gossip protocol (discovery and error detection)
    - Distributed key-value data store
    - Eventual consistency
  - CAP Theorem (discuss in a sec ..)

The Perfect Storm

- Large datasets, acceptance of alternatives, and dynamically-typed data has come together in a perfect storm
- Not a backlash/rebellion against RDBMS
- SQL is a rich query language that cannot be rivaled by the current list of NoSQL offerings

CAP Theorem

- Three properties of a system: consistency, availability and partitions
- You can have at most two of these three properties for any shared-data system
- To scale out, you have to partition. That leaves either consistency or availability to choose from
  - In almost all cases, you would choose availability over consistency

The CAP Theorem

The CAP Theorem

Once a writer has written, all readers will see that write

Consistency

- Two kinds of consistency:
  - strong consistency – ACID(Atomicity Consistency Isolation Durability)
  - weak consistency – BASE(Basically Available Soft-state Eventual consistency)

ACID Transactions

- A DBMS is expected to support "*ACID transactions*," processes that are:
  - *Atomic* : Either the whole process is done or none is.
  - *Consistent* : Database constraints are preserved.
  - *Isolated* : It appears to the user as if only one process executes at a time.
  - *Durable* : Effects of a process do not get lost if the system crashes.

Atomicity

- A real-world event either happens or does not happen
  - Student either registers or does not register
- Similarly, the system must ensure that either the corresponding transaction runs to completion or, if not, it has no effect at all
  - Not true of ordinary programs. A crash could leave files partially updated on recovery

Commit and Abort

- **If the transaction successfully completes it is said to commit**
  - **The system is responsible for ensuring that all changes to the database have been saved**
- **If the transaction does not successfully complete, it is said to abort**
  - **The system is responsible for undoing, or rolling back, all changes the transaction has made**

Database Consistency

- **Enterprise (Business) Rules limit the occurrence of certain real-world events**
  - **Student cannot register for a course if the current number of registrants equals the maximum allowed**
- **Correspondingly, allowable database states are restricted**

  cur_reg <= max_reg

- **These limitations are called (static) integrity constraints*:* assertions that must be satisfied by all database states (state invariants).**

Database Consistency
(state invariants)

- **Other static consistency requirements are related to the fact that the database might store the same information in different ways**
  - ***cur_reg* = |*list_of_registered_students*|**
  - **Such limitations are also expressed as integrity constraints**
- **Database is consistent if all static integrity constraints are satisfied**

Transaction Consistency

- **A consistent database state does not necessarily model the actual state of the enterprise**
  - **A deposit transaction that increments the balance by the wrong amount maintains the integrity constraint *balance* ≥ 0, but does not maintain the relation between the enterprise and database states**
- **A consistent transaction maintains database consistency and the correspondence between the database state and the enterprise state (implements its specification)**
  - **Specification of deposit transaction includes**

    $balance' = balance + amt_deposit$

    , **(*balance′* is the next value of *balance*)**

Dynamic Integrity Constraints
(transition invariants)

- **Some constraints restrict allowable state transitions**
  - **A transaction might transform the database from one consistent state to another, but the transition might not be permissible**
  - **Example: A letter grade in a course (A, B, C, D, F) cannot be changed to an incomplete (I)**
- **Dynamic constraints cannot be checked by examining the database state**

Transaction Consistency

- **Consistent transaction: if DB is in consistent state initially, when the transaction completes:**
  - **All static integrity constraints are satisfied (but constraints might be violated in intermediate states)**
    - **Can be checked by examining snapshot of database**
  - **New state satisfies specifications of transaction**
    - **Cannot be checked from database snapshot**
  - **No dynamic constraints have been violated**
    - **Cannot be checked from database snapshot**

Isolation

- **Serial Execution: transactions execute in sequence**
  - **Each one starts after the previous one completes.**
    - **Execution of one transaction is not affected by the operations of another since they do not overlap in time**
  - **The execution of each transaction is isolated from all others.**
- **If the initial database state and all transactions are consistent, then the final database state will be consistent and will accurately reflect the real-world state, *but***
- **Serial execution is inadequate from a performance perspective**

Isolation

- **Concurrent execution offers performance benefits:**
  - **A computer system has multiple resources capable of executing independently (*e.g.,* cpu's, I/O devices), *but***
  - **A transaction typically uses only one resource at a time**
  - **Hence, only concurrently executing transactions can make effective use of the system**
  - **Concurrently executing transactions yield interleaved schedules**

Concurrent Execution

Durability

- The system must ensure that once a transaction commits, its effect on the database state is not lost in spite of subsequent failures
  - Not true of ordinary programs. A media failure after a program successfully terminates could cause the file system to be restored to a state that preceded the program's execution

Implementing Durability

- Database stored redundantly on mass storage devices to protect against media failure
- Architecture of mass storage devices affects type of media failures that can be tolerated
- Related to Availability: extent to which a (possibly distributed) system can provide service despite failure
    - Non-stop DBMS (mirrored disks)
    - Recovery based DBMS (log)

Consistency Model

- A consistency model determines rules for visibility and apparent order of updates.
- For example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses.
  - Client B reads row X from node M
  - Does client B see the write from client A?
  - Consistency is a continuum with tradeoffs
  - For NoSQL, the answer would be: maybe
  - CAP Theorem states: Strict Consistency can't be achieved at the same time as availability and partition-tolerance.

Eventual Consistency

- When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
- For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service
- Known as BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency), as opposed to ACID

The CAP Theorem

System is available during software and hardware upgrades and node failures.

Availability

- Traditionally, thought of as the server/process available five 9's (99.999 %).
- However, for large node system, at almost any point in time there's a good chance that a node is either down or there is a network disruption among the nodes.
  - Want a system that is resilient in the face of network disruption

The CAP Theorem

A system can continue to operate in the presence of a network partitions.

The CAP Theorem

Theorem: You can have at most **two** of these properties for any shared-data system

What kinds of NoSQL

- NoSQL solutions fall into two major areas:
  - Key/Value or ‘the big hash table’.
    - Amazon S3 (Dynamo)
    - Voldemort
    - Scalaris
    - Memcached (in-memory key/value store)
    - Redis
  - Schema-less which comes in multiple flavors, column-based, document-based or graph-based.
    - Cassandra (column-based)
    - CouchDB (document-based)
    - MongoDB(document-based)
    - Neo4J (graph-based)
    - HBase (column-based)

Key/Value

Pros:

- very fast
- very scalable
- simple model
- able to distribute horizontally

Cons:

- many data structures (objects) can't be easily modeled as key value pairs

Schema-Less

Pros:

- Schema-less data model is richer than key/value pairs
- eventual consistency
- many are distributed
- still provide excellent performance and scalability

Cons:

- typically no ACID transactions or joins

Common Advantages

- Cheap, easy to implement (open source)
- Data are replicated to multiple nodes (therefore identical and fault-tolerant) and can be partitioned
  - Down nodes easily replaced
  - No single point of failure
- Easy to distribute
- Don't require a schema
- Can scale up and down
- Relax the data consistency requirement (CAP)

What am I giving up?

- joins
- group by
- order by
- ACID transactions
- SQL as a sometimes frustrating but still powerful query language
- easy integration with other applications that support SQL

Big Table and Hbase
(C+P)

Data Model

- A table in Bigtable is a sparse, distributed, persistent multidimensional sorted map
- Map indexed by a row key, column key, and a timestamp
-

  - Single row transactions only

Image Source: Chang et al., OSDI 2006

Rows and Columns

- Rows maintained in sorted lexicographic order
  - Applications can exploit this property for efficient row scans
  - Row ranges dynamically partitioned into tablets
- Columns grouped into column families
  - Column key = *family:qualifier*
  - Column families provide locality hints
  - Unbounded number of columns

Bigtable Building Blocks

- GFS
- Chubby
- SSTable

SSTable

- Basic building block of Bigtable
- Persistent, ordered immutable map from keys to values
  - Stored in GFS
- Sequence of blocks on disk plus an index for block lookup
  - Can be completely mapped into memory
- Supported operations:
  - Look up value associated with key
  - Iterate key/value pairs within a key range

Source: Graphic from slides by Erik Paulson

Tablet

- Dynamically partitioned range of rows
- Built from multiple SSTables

Source: Graphic from slides by Erik Paulson

Table

- Multiple tablets make up the table
- SSTables can be shared

Source: Graphic from slides by Erik Paulson

Architecture

- Client library
- Single master server
- Tablet servers

Bigtable Master

- Assigns tablets to tablet servers
- Detects addition and expiration of tablet servers
- Balances tablet server load
- Handles garbage collection
- Handles schema changes

Bigtable Tablet Servers

- Each tablet server manages a set of tablets
  - Typically between ten to a thousand tablets
  - Each 100-200 MB by default
- Handles read and write requests to the tablets
- Splits tablets that have grown too large

Tablet Location

Image Source: Chang et al., OSDI 2006

Tablet Assignment

- Master keeps track of:
  - Set of live tablet servers
  - Assignment of tablets to tablet servers
  - Unassigned tablets
- Each tablet is assigned to one tablet server at a time
  - Tablet server maintains an exclusive lock on a file in Chubby
  - Master monitors tablet servers and handles assignment
- Changes to tablet structure
  - Table creation/deletion (master initiated)
  - Tablet merging (master initiated)
  - Tablet splitting (tablet server initiated)

Tablet Serving

Image Source: Chang et al., OSDI 2006

Compactions

- Minor compaction
  - Converts the memtable into an SSTable
  - Reduces memory usage and log traffic on restart
- Merging compaction
  - Reads the contents of a few SSTables and the memtable, and writes out a new SSTable
  - Reduces number of SSTables
- Major compaction
  - Merging compaction that results in only one SSTable
  - No deletion records, only live data

Bigtable Applications

- Data source and data sink for MapReduce
- Google’s web crawl
- Google Earth
- Google Analytics

Lessons Learned

- Fault tolerance is hard
- Don’t add functionality before understanding its use
  - Single-row transactions appear to be sufficient
- Keep it simple!

HBase is an **open-source**, **distributed**, **column-oriented** database built on top of HDFS based on BigTable!

HBase is ..

- A distributed data store that can scale horizontally to 1,000s of commodity servers and petabytes of indexed storage.
- Designed to operate on top of the Hadoop distributed file system (HDFS) or Kosmos File System (KFS, aka Cloudstore) for scalability, fault tolerance, and high availability.

Benefits

- Distributed storage
- Table-like in data structure
  - multi-dimensional map
- High scalability
- High availability
- High performance

Backdrop

- Started toward by Chad Walters and Jim
- 2006.11
  - Google releases paper on BigTable
- 2007.2
  - Initial HBase prototype created as Hadoop contrib.
- 2007.10
  - First useable HBase
- 2008.1
  - Hadoop become Apache top-level project and HBase becomes subproject
- 2008.10~
  - HBase 0.18, 0.19 released

HBase Is Not …

- Tables have one primary index, the *row key*.
- No join operators.
- Scans and queries can select a subset of available columns, perhaps by using a wildcard.
- There are three types of lookups:
  - Fast lookup using row key and optional timestamp.
  - Full table scan
  - Range scan from region start to end.

HBase Is Not …(2)

- Limited atomicity and transaction support.
  - HBase supports multiple batched mutations of single rows only.
  - Data is unstructured and untyped.
- No accessed or manipulated via SQL.
  - Programmatic access via Java, REST, or Thrift APIs.
  - Scripting via JRuby.

Why Bigtable?

- Performance of RDBMS system is good for transaction processing but for very large scale analytic processing, the solutions are commercial, expensive, and specialized.
- Very large scale analytic processing
  - Big queries – typically range or table scans.
  - Big databases (100s of TB)

Why Bigtable? (2)

- Map reduce on Bigtable with optionally Cascading on top to support some relational algebras may be a cost effective solution.
- Sharding is not a solution to scale open source RDBMS platforms
  - Application specific
  - Labor intensive (re)partitionaing

Why HBase ?

- HBase is a Bigtable clone.
- It is open source
- It has a good community and promise for the future
- It is developed on top of and has good integration for the Hadoop platform, if you are using Hadoop already.
- It has a Cascading connector.

HBase benefits than RDBMS

- *No real indexes*
- *Automatic partitioning*
- *Scale linearly and automatically with new nodes*
- *Commodity hardware*
- *Fault tolerance*
- *Batch processing*

HBase: Part of Hadoop's Ecosystem

HBase is built on top of HDFS

HBase files are internally stored in HDFS

HBase vs. HDFS

- Both are distributed systems that scale to hundreds or thousands of nodes

- ***HDFS*** is good for batch processing (scans over big files)
  - Not good for record lookup
  - Not good for incremental addition of small batches
  - Not good for updates

HBase vs. HDFS (Cont'd)

- ***HBase*** is designed to efficiently address the above points
  - Fast record lookup
  - Support for record-level insertion
  - Support for updates (not in place)

- HBase updates are done by creating new versions of values

HBase vs. HDFS (Cont'd)

	Plain HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured storage	Do-it-yourself / TSV / SequenceFile / Avro / ?	Sparse column-family data model
Max data size	30+ PB	~1PB

If application has neither random reads or writes ➔ Stick to HDFS

HBase Data Model

HBase Data Model

- **HBase is based on Google's Bigtable model**
  - **Key-Value pairs**

HBase Logical View

Implicit PRIMARY KEY in RDBMS terms

Data is all `byte[]` in HBase

Different types of data separated into different "column families"

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

Different rows may have different sets of columns(table is *sparse*)

A single cell might have different values at different timestamps

Useful for *-To-Many mappings

HBase: Keys and Column Families

Each record is divided into *Column Families*

Each row has a *Key*

Figure 2. Census Data in Column Families

Each column family consists of one or more *Columns*

- **Key**
  - Byte array
  - Serves as the primary key for the table
  - Indexed far fast lookup
- **Column Family**
  - Has a name (string)
  - Contains one or more related columns
- **Column**
  - Belongs to one column family
  - Included inside the row
    - ***familyName:columnName***

Column family named "Contents"

Column family named "anchor"

Column named "apache.com"

Row key	Time Stamp	Column "contents:"	Column "anchor:"	
"com.apache.www"	t12	"<html> …"		
	t11	"<html> …"		
	t10		"anchor:apache.com"	"APACHE"
"com.cnn.www"	t15		"anchor:cnnsi.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
	t6	"<html> …"		
	t5	"<html> …"		
	t3	"<html> …"		

- **Version Number**
  - Unique within each key
  - By default→ System's timestamp
  - Data type is Long
- **Value (Cell)**
  - Byte array

Version number for each row

Row key	Time Stamp	Column "contents:"	Column "anchor:"	
"com.apache.www"	t12	"<html> …"		value
	t11	"<html> …"		
	t10		"anchor:apache.com"	"APACHE"
"com.cnn.www"	t15		"anchor:cnnsi.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
	t6	"<html> …"		
	t5	"<html> …"		
	t3	"<html> …"		

Notes on Data Model

- HBase schema consists of several ***Tables***
- Each table consists of a set of ***Column Families***
  - Columns are not part of the schema
- HBase has ***Dynamic Columns***
  - Because column names are encoded inside the cells
  - Different cells can have different columns

"Roles" column family has different columns in different cells

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

Notes on Data Model (Cont'd)

- The ***version number*** can be user-supplied
  - Even does not have to be inserted in increasing order
  - Version number are unique within each key
- Table can be very sparse
  - Many cells are empty
- ***Keys*** are indexed as the primary key

Has two columns
[cnnsi.com & my.look.ca]

Row Key	Time Stamp	ColumnFamily `contents`	ColumnFamily `anchor`
"com.cnn.www"	t9		`anchor:cnnsi.com` = "CNN"
"com.cnn.www"	t8		`anchor:my.look.ca` = "CNN.com"
"com.cnn.www"	t6	`contents:html` = "<html>..."	
"com.cnn.www"	t5	`contents:html` = "<html>..."	
"com.cnn.www"	t3	`contents:html` = "<html>..."	

HBase Physical Model

HBase Physical Model

- Each column family is stored in a separate file (called ***HTables***)
- Key & Version numbers are replicated with each column family
- Empty cells are not stored

HBase maintains a multi-level index on values:
<key, column family, column name, timestamp>

Table 5.3. ColumnFamily `contents`

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	`contents:html` = "<html>..."
"com.cnn.www"	t5	`contents:html` = "<html>..."
"com.cnn.www"	t3	`contents:html` = "<html>..."

Table 5.2. ColumnFamily `anchor`

Row Key	Time Stamp	Column Family `anchor`
"com.cnn.www"	t9	`anchor:cnnsi.com` = "CNN"
"com.cnn.www"	t8	`anchor:my.look.ca` = "CNN.com"

Example

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

info Column Family

Row key	Column key	Timestamp	Cell value
cutting	info:height	1273516197868	9ft
cutting	info:state	1043871824184	CA
tlipcon	info:height	1273878447049	5ft7
tlipcon	info:state	1273616297446	CA

roles Column Family

Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

Sorted on disk by Row key, Col key, descending timestamp

Milliseconds since unix epoch

cloudera

Column Families

- Different sets of columns may have different properties and access patterns
- Configurable by column family:
  - Compression (none, gzip, LZO)
  - Version retention policies
  - Cache priority
- CFs stored separately on disk: access one without wasting IO on the other.

HBase Regions

- Each HTable (column family) is partitioned horizontally into ***regions***
  - Regions are counterpart to HDFS blocks

Table 5.3. ColumnFamily `contents`

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	`contents:html` = "<html>..."
"com.cnn.www"	t5	`contents:html` = "<html>..."
"com.cnn.www"	t3	`contents:html` = "<html>..."

Each will be one region

HBase Architecture

Three Major Components

- The HBaseMaster
  - One master
- The HRegionServer
  - Many region servers
- The HBase client

HBase Components

- **Region**
  - A subset of a table's rows, like horizontal range partitioning
  - Automatically done
- **RegionServer (many slaves)**
  - Manages data regions
  - Serves data for reads and writes (***using a log***)
- **Master**
  - Responsible for coordinating the slaves
  - Assigns regions, detects failures
  - Admin functions

Big Picture

Client
Zookeeper
HMaster
HBase
HRegionServer
HRegion
HLog
Store
MemStore
StoreFile
HFile
DFS Client
Hadoop
DataNode

ZooKeeper

- HBase depends on ZooKeeper
- By default HBase manages the ZooKeeper instance
  - E.g., starts and stops ZooKeeper
- HMaster and HRegionServers register themselves with ZooKeeper

Creating a Table

```
HBaseAdmin admin= new HBaseAdmin(config);
HColumnDescriptor []column;
column= new HColumnDescriptor[2];
column[0]=new
   HColumnDescriptor("columnFamily1:");
column[1]=new
   HColumnDescriptor("columnFamily2:");
HTableDescriptor desc= new HTableDescriptor(Bytes.toBytes("MyTable"));
desc.addFamily(column[0]);
desc.addFamily(column[1]);
admin.createTable(desc);
```

Operations On Regions: **Get()**

- Given a key → return corresponding record
- For each value return the highest version

```
Get get = new Get(Bytes.toBytes("row1"));
Result r = htable.get(get);
5.8.1.2. Default Get Example ue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
```

- Can control the number of versions you want

```
Get get = new Get(Bytes.toBytes("row1"));
get.setMaxVersions(3);  // will return last 3 versions of row
Result r = htable.get(get);
byte[] b = r.getValue(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns current version of value
List<KeyValue> kv = r.getColumn(Bytes.toBytes("cf"), Bytes.toBytes("attr"));  // returns all versions of
```

Operations On Regions: **Scan()**

```
HTable htable = ...      // instantiate HTable

Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("cf"),Bytes.toBytes("attr"));
scan.setStartRow( Bytes.toBytes("row"));                    // start key is inclusive
scan.setStopRow( Bytes.toBytes("row" +  (char)0));  // stop key is exclusive
ResultScanner rs = htable.getScanner(scan);
try {
  for (Result r = rs.next(); r != null; r = rs.next()) {
  // process result...
} finally {
  rs.close();  // always close the ResultScanner!
}
```

Get()

Select value from table where key= ‘com.apache.www’ AND label= ‘anchor:apache.com’

Row key	Time Stamp	Column “anchor:”	
“com.apache.www”	t12		
	t11		
	t10	“anchor:apache.com”	**“APACHE”**
“com.cnn.www”	t9	“anchor:cnnsi.com”	“CNN”
	t8	“anchor:my.look.ca”	“CNN.com”
	t6		
	t5		
	t3		

Scan()

Select value from table where
anchor= ‘cnnsi.com’

Row key	Time Stamp	Column “anchor:”	
“com.apache.www”	t12		
	t11		
	t10	“anchor:apache.com”	“APACHE”
“com.cnn.www”	t9	“anchor:cnnsi.com”	“CNN”
	t8	“anchor:my.look.ca”	“CNN.com”
	t6		
	t5		
	t3		

Operations On Regions: **Put()**

- Insert a new record (with a new key), Or
- Insert a record for an existing key

Implicit version number (timestamp)

```
Put put = new Put(Bytes.toBytes(row));
put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), Bytes.toBytes( data));
htable.put(put);
```

Explicit version number

```
Put put = new Put( Bytes.toBytes(row));
long explicitTimeInMs = 555;  // just an example
put.add(Bytes.toBytes("cf"), Bytes.toBytes("attr1"), explicitTimeInMs, Bytes.toBytes(data));
htable.put(put);
```

Operations On Regions: **Delete()**

- Marking table cells as deleted
- **Multiple levels**
  - Can mark an entire column family as deleted
  - Can make all column families of a given row as deleted

- **All operations are logged by the RegionServers**
- **The log is flushed periodically**

HBase: **Joins**

- HBase does not support joins
- Can be done in the application layer
  - Using scan() and get() operations

Altering a Table

```
Configuration config = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(conf);
String table = "myTable";

admin.disableTable(table);

HColumnDescriptor cf1 = ...;
admin.addColumn(table, cf1);       // adding new ColumnFamily
HColumnDescriptor cf2 = ...;
admin.modifyColumn(table, cf2);     // modifying existing ColumnFamily

admin.enableTable(table);
```

Disable the table before changing the schema

6.1. Schema Creation

Logging Operations

HBase Deployment

HBase vs. HDFS

	Plain HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured storage	Do-it-yourself / TSV / SequenceFile / Avro / ?	Sparse column-family data model
Max data size	30+ PB	~1PB

HBase vs. RDBMS

	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorization	Work in progress
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of queries/second

When to use HBase

- You need random write, random read, or both (*but not neither*)
- You need to do many thousands of operations per second on multiple TB of data
- Your access patterns are well-known and simple

Hive and Pig

Need for High-Level Languages

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Not everyone wants to (or can) write Java code
- Solution: develop higher-level data processing languages
  - Hive: HQL is like SQL
  - Pig: Pig Latin is a bit like Perl

Hive and Pig

- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source
- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow langua
  - Developed by Yahoo!, now open source
  - Roughly 1/3 of all Yahoo! internal jobs

- Common idea:
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language “compiles down” to Hadoop jobs

Hive: Background

- Started at Facebook
- Data was collected by nightly cron jobs into Oracle DB
- “ETL” via hand-coded python
- Grew from 10s of GBs (2006) to 1 TB/day new data (2007), now 10x that

Source: cc-licensed slide by Cloudera

Hive Components

- Shell: allows interactive queries
- Driver: session handles, fetch, execute
- Compiler: parse, plan, optimize
- Execution engine: DAG of stages (MR, HDFS, metadata)
- Metastore: schema, location in HDFS, SerDe

Source: cc-licensed slide by Cloudera

Data Model

- Tables
  - Typed columns (int, float, string, boolean)
  - Also, list: map (for JSON-like data)
- Partitions
  - For example, range-partition tables by date
- Buckets
  - Hash partitions within ranges (useful for sampling, join optimization)

Source: cc-licensed slide by Cloudera

Metastore

- Database: namespace containing a set of tables
- Holds table definitions (column types, physical layout)
- Holds partitioning information
- Can be stored in Derby, MySQL, and many other relational databases

Source: cc-licensed slide by Cloudera

Physical Layout

- Warehouse directory in HDFS
  - E.g., /user/hive/warehouse
- Tables stored in subdirectories of warehouse
  - Partitions form subdirectories of tables
- Actual data stored in flat files
  - Control char-delimited text, or SequenceFiles
  - With custom SerDe, can use arbitrary format

Source: cc-licensed slide by Cloudera

Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

the	25848	62394
I	23031	8854
and	19671	38985
to	18038	13526
of	16700	34654
a	14170	8057
you	12702	2720
my	11297	4135
in	10797	12445
is	8882	6884

Source: Material drawn from Cloudera training VM

Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

(Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (. (TOK_TABLE_OR_COL s)
word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT
(TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (.
(TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k)
freq) 1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```

(one or more of MapReduce jobs)

Hive: Behind the Scenes

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        s
          TableScan
            alias: s
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                    expr: word
                    type: string
                sort order: +
                Map-reduce partition columns:
                    expr: word
                    type: string
                tag: 0
                value expressions:
                    expr: freq
                    type: int
                    expr: word
                    type: string
        k
          TableScan
            alias: k
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                    expr: word
                    type: string
                sort order: +
                Map-reduce partition columns:
                    expr: word
                    type: string
                tag: 1
                value expressions:
                    expr: freq
                    type: int
      Reduce Operator Tree:
        Join Operator
          condition map:
              Inner Join 0 to 1
          condition expressions:
            0 {VALUE._col0} {VALUE._col1}
            1 {VALUE._col0}
          outputColumnNames: _col0, _col1, _col2
          Filter Operator
            predicate:
                expr: ((_col0 >= 1) and (_col2 >= 1))
                type: boolean
            Select Operator
              expressions:
                  expr: _col1
                  type: string
                  expr: _col0
                  type: int
                  expr: _col2
                  type: int
              outputColumnNames: _col0, _col1, _col2
              File Output Operator
                compressed: false
                GlobalTableId: 0
                table:
                    input format: org.apache.hadoop.mapred.SequenceFileInputFormat
                    output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat

  Stage: Stage-2
    Map Reduce
      Alias -> Map Operator Tree:
        hdfs://localhost:8022/tmp/hive-training/364214370/10002
          Reduce Output Operator
            key expressions:
                expr: _col1
                type: int
            sort order: -
            tag: -1
            value expressions:
                expr: _col0
                type: string
                expr: _col1
                type: int
                expr: _col2
                type: int
      Reduce Operator Tree:
        Extract
          Limit
            File Output Operator
              compressed: false
              GlobalTableId: 0
              table:
                  input format: org.apache.hadoop.mapred.TextInputFormat
                  output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

  Stage: Stage-0
    Fetch Operator
      limit: 10
```

Example Data Analysis Task

Find users who tend to visit “good” pages.

Visits

user	url	time
Amy	www.cnn.com	8:00
Amy	www.crap.com	8:05
Amy	www.myblog.com	10:00
Amy	www.flickr.com	10:05
Fred	cnn.com/index.htm	12:00

⋮

Pages

url	pagerank
www.cnn.com	0.9
www.flickr.com	0.9
www.myblog.com	0.7
www.crap.com	0.2

⋮

Pig Slides adapted from Olston et al.

Conceptual Dataflow

Pig Slides adapted from Olston et al.

System-Level Dataflow

Pig Slides adapted from Olston et al.

MapReduce Code

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.a    pache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
imp ort org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobC                ontrol;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.sub            string(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1            " + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(             firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so w                e know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.to            String();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                    }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputColle         ctor<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', first                    Comma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
             oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text ke        y,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            wh      ile (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        i   mplements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter)           throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public      void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count        < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.se    tJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/  user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.s    etJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.add         InputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(           MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMap             per.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.se        tOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRE            xample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFi                leOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
           FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
         top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputF                    ormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
           FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top                100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

Pig Slides adapted from Olston et al.

Pig Latin Script

```
Visits = load    '/data/visits' as (user, url, time);
Visits = foreach Visits generate user, Canonicalize(url), time;

Pages = load    '/data/pages' as (url, pagerank);

VP = join    Visits by url, Pages by url;
UserVisits = group   VP by user;
UserPageranks = foreach UserVisits generate user, AVG(VP.pagerank) as avgpr;
GoodUsers = filter  UserPageranks by avgpr > '0.5';

store   GoodUsers into '/data/good_users';
```

Pig Slides adapted from Olston et al.

Java vs. Pig Latin

1/20 the lines of code

1/16 the development time

Performance on par with raw Hadoop!

Pig Slides adapted from Olston et al.

Pig takes care of…

- Schema and type checking
- Translating into efficient physical dataflow
  - (i.e., sequence of one or more MapReduce jobs)
- Exploiting data reduction opportunities
  - (e.g., early partial aggregation via a combiner)
- Executing the system-level dataflow
  - (i.e., running the MapReduce jobs)
- Tracking progress, errors, etc.

Hive + HBase?

Integration

- Reasons to use Hive on HBase:
  - A lot of data sitting in HBase due to its usage in a real-time environment, but never used for analysis
  - Give access to data in HBase usually only queried through MapReduce to people that don't code (business analysts)
  - When needing a more flexible storage solution, so that rows can be updated live by either a Hive job or an application and can be seen immediately to the other

- Reasons not to do it:
  - Run SQL queries on HBase to answer live user requests (it's still a MR job)
  - Hoping to see interoperability with other SQL analytics systems

Integration

- How it works:
  - Hive can use tables that already exist in HBase or manage its own ones, but they still all reside in the same HBase instance

Integration

- How it works:
  - When using an already existing table, defined as EXTERNAL, you can create multiple Hive tables that point to it

Integration

- How it works:
  - Columns are mapped however you want, changing names and giving types

Integration

- Drawbacks (that can be fixed with brain juice):
  - Binary keys and values (like integers represented on 4 bytes) aren't supported since Hive prefers string representations, HIVE-1634
  - Compound row keys aren't supported, there's no way of using multiple parts of a key as different "fields"
  - This means that concatenated binary row keys are completely unusable, which is what people often use for HBase
  - Filters are done at Hive level instead of being pushed to the region servers
  - Partitions aren't supported

Data Flows

- Data is being generated all over the place:
  - Apache logs
  - Application logs
  - MySQL clusters
  - HBase clusters

Data Flows

- Moving application log files

Data Flows

- Moving MySQL data

Data Flows

- Moving HBase data

Use Cases

- Front-end engineers
  - They need some statistics regarding their latest product
- Research engineers
  - Ad-hoc queries on user data to validate some assumptions
  - Generating statistics about recommendation quality
- Business analysts
  - Statistics on growth and activity
  - Effectiveness of advertiser campaigns
  - Users' behavior VS past activities to determine, for example, why certain groups react better to email communications
  - Ad-hoc queries on stumbling behaviors of slices of the user base

Use Cases

- Using a simple table in HBase:

```
CREATE EXTERNAL TABLE blocked_users(
 userid INT,
 blockee INT,
 blocker INT,
 created BIGINT)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler’
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
 ":key,f:blockee,f:blocker,f:created")
TBLPROPERTIES("hbase.table.name" = "m2h_repl-userdb.stumble.blocked_users");
```

HBase is a special case here, it has a unique row key map with :key
Not all the columns in the table need to be mapped

Use Cases

- Using a complicated table in HBase:

```
CREATE EXTERNAL TABLE ratings_hbase(
 userid INT,
 created BIGINT,
 urlid INT,
 rating INT,
 topic INT,
 modified BIGINT)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler’
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
 ":key#b@0,:key#b@1,:key#b@2,default:rating#b,default:topic#b,default:modified#b")
TBLPROPERTIES("hbase.table.name" = "ratings_by_userid");
```

#b means binary, @ means position in composite key (SU-specific hack)

Graph Databases

NEO4J (Graphbase)

- A graph is a collection nodes (things) and edges (relationships) that connect pairs of nodes.

- Attach properties (key-value pairs) on nodes and relationships

- Relationships connect two nodes and both nodes and relationships can hold an arbitrary amount of key-value pairs.

- A graph database can be thought of as a key-value store, with full support for relationships.

- http://neo4j.org/

NEO4J

2.1.1. A Graph contains Nodes and Relationships

"A Graph —records data in→ Nodes —which have→ Properties"

The simplest possible graph is a single Node, a record that has named values referred to as Properties. A Node could start with a single Property and grow to a few million, though that can get a little awkward. At some point it makes sense to distribute the data into multiple nodes, organized with explicit Relationships.

NEO4J

2.1.2. Relationships organize the Graph

"Nodes —are organized by→ Relationships —which also have→ Properties"

Relationships organize Nodes into arbitrary structures, allowing a Graph to resemble a List, a Tree, a Map, or a compound Entity - any of which can be combined into yet more complex, richly inter-connected structures.

NEO4J

2.1.3. Query a Graph with a Traversal

"A Traversal —navigates→ a Graph; it —identifies→ Paths —which order→ Nodes"

A Traversal is how you query a Graph, navigating from starting Nodes to related Nodes according to an algorithm, finding answers to questions like "what music do my friends like that I don't yet own," or "if this power supply goes down, what web services are affected?"

NEO4J

2.1.4. Indexes look-up Nodes or Relationships

"An Index —maps from→ Properties —to either→ Nodes or Relationships"

Often, you want to find a specific Node or Relationship according to a Property it has. Rather than traversing the entire graph, use an Index to perform a look-up, for questions like "find the Account for username master-of-graphs."

NEO4J

2.1.5. Neo4j is a Graph Database

"A Graph Database —manages a→ Graph and —also manages related→ Indexes"

Neo4j is a commercially supported open-source graph database. It was designed and built from the ground-up to be a reliable database, optimized for graph structures instead of tables. Working with Neo4j, your application gets all the expressiveness of a graph, with all the dependability you expect out of a database.

NEO4J

Properties

Properties are key-value pairs where the key is a string. Property values can be either a primitive or an array of one primitive type. For example `String`, `int` and `int[]` values are valid for properties.

Note

`null` is not a valid property value. Nulls can instead be modeled by the absence of a key.

NEO4J Features

- Dual license: open source and commercial
- Well suited for many web use cases such as tagging, metadata annotations, social networks, wikis and other network-shaped or hierarchical data sets
- Intuitive graph-oriented model for data representation. Instead of static and rigid tables, rows and columns, you work with a flexible graph network consisting of nodes, relationships and properties.
- Neo4j offers performance improvements on the order of 1000x or more compared to relational DBs.
- A disk-based, native storage manager completely optimized for storing graph structures for maximum performance and scalability
- Massive scalability. Neo4j can handle graphs of several billion nodes/relationships/properties on a single machine and can be sharded to scale out across multiple machines
- Fully transactional like a real database
- Neo4j traverses depths of 1000 levels and beyond at millisecond speed. (many orders of magnitude faster than relational systems)

Spark SQL: Relational Data Processing in Spark

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

SIGMOD 2015 – Melbourne, Australia

Presented by Doris Xin

Based on slides provided by M. Armbrust

Challenges and Solutions

Challenges

- Perform ETL to and from various (semi- or unstructured) data sources
- Perform advanced analytics (e.g. machine learning, graph processing) that are hard to express in relational systems.

Solutions

- A *DataFrame* API that can perform relational operations on both external data sources and Spark's built-in RDDs.
- A highly extensible optimizer, *Catalyst*, that uses features of Scala to add composable rule, control code gen., and define extensions.

What is Apache Spark?

Fast and general cluster computing system, interoperable with Hadoop, included in all major distros

Improves efficiency through:

- In-memory computing primitives
- General computation graphs

→ Up to 100× faster (2-10× on disk)

Improves usability through:

- Rich APIs in Scala, Java, Python
- Interactive shell

→ 2-5× less code

Spark Model

Write programs in terms of transformations on distributed datasets

Resilient Distributed Datasets (RDDs)

- Collections of objects that can be stored in memory or disk across a cluster
- Parallel functional transformations (map, filter, …)
- Automatically rebuilt on failure

On-Disk Sort Record:

Time to sort 100TB

2013 Record: Hadoop

2100 machines

72 minutes

2014 Record: Spark

207 machines

23 minutes

Also sorted 1PB in 4 hours

Source: Daytona GraySort benchmark, sortbenchmark.org

Powerful Stack – Agile Development

non-test, non-example source lines

databricks™

Powerful Stack – Agile Development

non-test, non-example source lines

databricks™

Powerful Stack – Agile Development

non-test, non-example source lines

databricks™

Powerful Stack – Agile Development

non-test, non-example source lines

databricks™

Powerful Stack – Agile Development

non-test, non-example source lines

databricks™

About Spark SQL

Spark SQL

- Part of the core distribution since Spark 1.0 (April 2014)

About Spark SQL

Spark SQL

- Part of the core distribution since Spark 1.0 (April 2014)
- Runs SQL / HiveQL queries, optionally alongside or replacing existing Hive deployments

```
SELECT COUNT(*)
FROM hiveTable
WHERE hive_udf(data)
```

Improvement upon Existing Art

Engine does not understand the structure of the data in RDDs or the semantics of user functions → limited optimization.

Can only be used to query external data in Hive catalog → limited data sources

Can only be invoked via SQL string from Spark→ error prone

Hive optimizer tailored for MapReduce → difficult to extend

Programming Interface

Figure 1: Interfaces to Spark SQL, and interaction with Spark.

```
ctx = new HiveContext()
users = ctx.table("users")
young = users.where(users("age") < 21)
println(young.count())
```

- A distributed collection of rows with the same schema (RDDs suffer from type erasure)
- Can be constructed from external data sources or RDDs into essentially an RDD of Row objects (SchemaRDDs as of Spark < 1.3)
- Supports relational operators (e.g. *where*, *groupby*) as well as Spark operations.
- Evaluated lazily → unmaterialized *logical* plan

Data Model

- Nested data model
- Supports both primitive SQL types (boolean, integer, double, decimal, string, data, ti mestamp) and complex types (structs, arrays, maps, and unions); also user defined types.
- First class support for complex data types

DataFrame Operations

- Relational operations (select, where, join, groupBy) via a DSL
- Operators take *expression* objects
- Operators build up an abstract syntax tree (AST), which is then optimized by *Catalyst*.

- Alternat
traditior

```
employees
  .join(dept, employees("deptId") === dept("id"))
  .where(employees("gender") === "female")
  .groupBy(dept("id"), dept("name"))
  .agg(count("name"))
```

```
users.where(users("age") < 21)
     .registerTempTable("young")
ctx.sql("SELECT count(*), avg(age) FROM young")
```

Advantages over Relational Query Languages

- Holistic optimization across functions composed in different languages.
- Control structures (e.g. *if*, *for*)
- Logical plan analyzed *eagerly* → identify code errors associated with data *schema* issues on the fly.

Querying Native Datasets

- Infer column names and types directly from data objects (via reflection in Java and Scala and data sampling in Python, which is dynamically typed)

- Nativ
```
case class User(name: String, age: Int)
```
expensive data format transformation.
- Benefits:
  - Run relational operations on existing Spark programs.
  - Combine RDDs with external structured data

Columnar storage with *hot* columns cached in memory

User-Defined Functions (UDFs)

- Easy extension of limited operations supported.
- Allows inline registration of UDFs
  - Compare with Pig, which requires the UDF to be written in a Java package that's loaded into the Pig script.
- Can be defined on simple data types or entire tables.
- UDFs available to other interfaces after registration

```
val model: LogisticRegressionModel = ...

ctx.udf.register("predict",
  (x: Float, y: Float) => model.predict(Vector(x, y)))

ctx.sql("SELECT predict(age, weight) FROM users")
```

Catalyst

```
tree.transform {
  case Add(Literal(c1), Literal(c2)) => Literal(c1+c2)
}
```

x + (1 + 2)

x + 3

Prior Work: Optimizer Generators

Volcano / Cascades:

- Create a custom language for expressing rules that rewrite trees of relational operators.
- Build a compiler that generates executable code for these rules.

Cons: Developers need to learn this custom language. Language might not be powerful enough.

Catalyst Rules

- *Pattern matching* functions that transform subtrees into specific structures.
  - *Partial function*—skip over subtrees that do not match → no need to modify existing rules when adding new types of operators.
- Multiple patterns in the same *transform* call.
- May take multiple *batches* to reach a *fixed point*.
- *transform* can contain arbitrary Scala code.

Plan Optimization & Execution

DataFrames and SQL share the same optimization/execution pipeline

Plan Optimization & Execution

DataFrames and SQL share the same optimization/execution pipeline

Analysis

SELECT *col* FROM *sales*

- An attribute is *unresolved* if its type is not known or it's not matched to an input table.
- To resolve attributes:
  - Look up relations by name from the catalog.
  - Map named attributes to the input provided given operator's children.
  - UID for references to the same value
  - Propagate and coerce types through expressions (e.g. 1 + *col*)

Plan Optimization & Execution

DataFrames and SQL share the same optimization/execution pipeline

Plan Optimization & Execution

DataFrames and SQL share the same optimization/execution pipeline

Logical Optimization

- Applies standard rule-based optimization (constant folding, predicate-pushdown, projection pruning, null propagation, boolean expression simplification, etc)
- 800LOC

```
object DecimalAggregates extends Rule[LogicalPlan] {
  /** Maximum number of decimal digits in a Long */
  val MAX_LONG_DIGITS = 18

  def apply(plan: LogicalPlan): LogicalPlan = {
    plan transformAllExpressions {
      case Sum(e @ DecimalType.Expression(prec, scale))
              if prec + 10 <= MAX_LONG_DIGITS =>
        MakeDecimal(Sum(LongValue(e)), prec + 10, scale)
    }
}
```

Plan Optimization & Execution

DataFrames and SQL share the same optimization/execution pipeline

Plan Optimization & Execution

e.g. Pipeline projections
and filters into a single *map*

DataFrames and SQL share the same optimization/execution pipeline

```python
def add_demographics(events):
    u = sqlCtx.table("users")                    # Load partitioned Hive table
    events \
      .join(u, events.user_id == u.user_id) \    # Join on user_id
      .withColumn("city", zipToCity(df.zip))     # Run udf to add city column
```

```python
events = add_demographics(sqlCtx.load("/data/events", "parquet"))
training_data = events.where(events.city == "Melbourne").select(events.timestamp).collect()
```

An Example Catalyst Transformation

1. Find filters on top of projections.
2. Check that the filter can be evaluated without the result of the project.
3. If so, switch the operators.

Plan Optimization & Execution

DataFrames and SQL share the same optimization/execution pipeline

Code Generation

```
def compile(node: Node): AST = node match {
  case Literal(value) => q"$value"
  case Attribute(name) => q"row.get($name)"
  case Add(left, right) =>
    q"${compile(left)} + ${compile(right)}"
}
```

Figure 4: A comparision of the performance evaluating the expresion x+x+x, where x is an integer, 1 billion times.

- Relies on Scala's *quasiquotes* to simplify code gen.
- Catalyst transforms a SQL tree into an abstract syntax tree (AST) for Scala code to eval expr and generate code
- 700LOC

Extensions

Data Sources

- must implement a *createRelation* function that takes a set of key-value params and returns a *BaseRelation* object.
- E.g. CSV, Avro, Parquet, JDBC

User-Defined Types (UDTs)

- Map user-defined types to structures composed of Catalyst's built-in types.

```
class PointUDT extends UserDefinedType[Point] {
  def dataType = StructType(Seq( // Our native structure
    StructField("x", DoubleType),
    StructField("y", DoubleType)
  ))
  def serialize(p: Point) = Row(p.x, p.y)
  def deserialize(r: Row) =
    Point(r.getDouble(0), r.getDouble(1))
}
```

Advanced Analytics Features

Schema Inference for Semistructured Data

JSON

- Automatically infers schema from a set of records, in one pass or sample
- A tree of STRUCT types, each of which may contain atoms, arrays, or other STRUCTs.
- Find the most appropriate type for a field based on all data observed in that column. Determine array element types in the same way.
- Merge schemata of single records in one *reduce* operation.
- Same trick for Python typing

```
{
  "text": "This is a tweet about #Spark",
  "tags": ["#Spark"],
  "loc": {"lat": 45.1, "long": 90}
}

{
  "text": "This is another tweet",
  "tags": [],
  "loc": {"lat": 39, "long": 88.5}
}

{
  "text": "A #tweet without #location",
  "tags": ["#tweet", "#location"]
}
```

```
text STRING NOT NULL,
tags ARRAY<STRING NOT NULL> NOT NULL,
loc STRUCT<lat FLOAT NOT NULL, long FLOAT NOT NULL>
```

Spark MLlib Pipelines

```
tokenizer = Tokenizer(inputCol="text", outputCol="words”)
hashingTF = HashingTF(inputCol="words", outputCol="features”)
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

df = sqlCtx.load("/path/to/data")
model = pipeline.fit(df)
```

110GB of data after columnar compression with Parquet

Figure 9: Performance of an aggregation written using the native Spark Python and Scala APIs versus the DataFrame API.

Figure 10: Performance of a two-stage pipeline written as a separate Spark SQL query and Spark job (above) and an integrated DataFrame job (below).

Research Transformations: Generalized Online Aggregation

Full Prototype: 3000 lines of code

SIGMOD Demo A

Research Transformation: Genomics

Recognize range joins and use interval trees.

```
SELECT * FROM a JOIN b
WHERE a.start < a.end
  AND b.start < b.end
  AND a.start < b.start
  AND b.start < a.end
```

amplab UC BERKELEY

Future Work: Project Tungsten

Overcome JVM limitations:

- **Memory Management and Binary Processing**: leveraging application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection
- **Cache-aware computation**: algorithms and data structures to exploit memory hierarchy
- **Code generation**: using code generation to exploit modern compilers and CPUs

Questions?

The not-so-secret truth...

is about more than SQL.

Spark SQL: Declarative BigData Processing

Let Developers Create and Run Spark Programs Faster:

- Write less code
- Read less data
- Let the optimizer do the hard work

DataFrame

noun – [dey-tuh-freym]

1. A distributed collection of rows organized into named columns.
2. An abstraction for selecting, filtering, aggregating and plotting structured data (*cf. R, Pandas*).
3. Archaic: Previously SchemaRDD (*cf. Spark < 1.3*).

Write Less Code: Compute an Average

```
private IntWritable one =
  new IntWritable(1)
private IntWritable output =
  new IntWritable()
proctected void map(
    LongWritable key,
    Text value,
    Context context) {
  String[] fields = value.split("\t")
  output.set(Integer.parseInt(fields[1]))
  context.write(one, output)
}

IntWritable one = new IntWritable(1)
DoubleWritable average = new DoubleWritable()

protected void reduce(
    IntWritable key,
    Iterable<IntWritable> values,
    Context context) {
  int sum = 0
  int count = 0
  for(IntWritable value : values) {
     sum += value.get()
     count++
    }
  average.set(sum / (double) count)
  context.Write(key, average)
}
```

```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [x.[1], 1])) \
   .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
   .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
   .collect()
```

Write Less Code: Compute an Average

Using RDDs

```
data = sc.textFile(...).split("\t")
data.map(lambda x: (x[0], [int(x[1]), 1])) \
   .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
   .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
   .collect()
```

Using SQL

```
SELECT name, avg(age)
FROM people
GROUP BY name
```

Using Pig

```
P = load '/people' as (name, name);
G = group P by name;
R = foreach G generate … AVG(G.age);
```

Using DataFrames

```
sqlCtx.table("people") \
   .groupBy("name") \
   .agg("name", avg("age")) \
   .collect()
```

Seamlessly Integrated: RDDs

Internally, DataFrame execution is done with Spark RDDs making interoperation with outside sources and custom algorithms easy.

External Input

```
def buildScan(
    requiredColumns: Array[String],
    filters: Array[Filter]):
RDD[Row]
```

Custom Processing

```
queryResult.rdd.mapPartitions { iter =>

  … Your code here …

}
```

Extensible Input & Output

Spark's Data Source API allows optimizations like column pruning and filter pushdown into custom data sources.

Built-In

{ JSON }

JDBC

Parquet

HIVE

MySQL

PostgreSQL

HDFS

amazon web services S3

H2

External

AVRO

CSV

dBase

APACHE HBASE

elasticsearch.

cassandra

amazon web services Amazon Redshift

memsql

and more…

Seamlessly Integrated

Embedding in a full programming language makes UDFs trivial and allows composition using functions.

```
zipToCity = udf(lambda city: <custom logic here>)

def add_demographics(events):
   u = sqlCtx.table("users")
   events \
     .join(u, events.user_id == u.user_id) \
     .withColumn("city", zipToCity(df.zip))
```

Takes and returns a DataFrame

databricks

Spark MLlib Pipelines

```
tokenizer = Tokenizer(inputCol="text", outputCol="words”)
hashingTF = HashingTF(inputCol="words", outputCol="features”)
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

df = sqlCtx.load("/path/to/data")
model = pipeline.fit(df)
```