CSE 232A
Graduate Database Systems

Arun Kumar

Topic 7: Parallel RDBMSs and Dataflow Systems

Chapters 22 of Cow Book
Outline

❖ Parallel RDBMSs and Cloud-Native RDBMSs
❖ Beyond RDBMSs: A Brief History
❖ “Big Data” Systems
Parallel DBMSs: Motivation

❖ **Scalability**: Database is too large for a single node’s disk
❖ **Performance**: Exploit multiple cores/disks/nodes
❖ … while maintaining almost all other benefits of (R)DBMSs!
Three Paradigms of Parallelism

- **Shared-Disk Parallelism**
  - Symmetric Multi-Processing (SMP)

- **Shared-Memory Parallelism**
  - Contention
  - Interconnect
  - Data/Partitioned Parallelism

- **Shared-Nothing Parallelism**
  - Interconnect
  - Massively Parallel Processing (MPP)
Shared-Nothing Parallelism

❖ Followed by almost all parallel RDBMSs (and “Big Data” sys.)
❖ 1 master node orchestrates multiple worker nodes
❖ Need partitioned parallel implementation algorithms for relational op implementations and query proc.; modify QO

Q: If we give 10 workers (CPUs/nodes) for processing a query in parallel, will its runtime go down by a factor of 10?

It depends!

(Access patterns of the query’s operators, communication of intermediate data, relative startup overhead, etc.)
Shared-Nothing Parallelism

Q: Is superlinear speedup/scaleup possible?
Shared-Nothing Parallelism: Outline

❖ Data Partitioning
❖ Parallel Operator Implementations
❖ Parallel Query Optimization
❖ Parallel vs “Distributed” DBMSs
Data Partitioning

❖ A part of ETL (Extract-Transform-Load) for database
❖ Typically, record-wise/horizontal partitioning (aka “sharding”)
❖ Three common schemes (given k machines):
  ❖ Round-robin: assign tuple i to machine i MOD k
  ❖ Hashing-based: needs partitioning attribute(s)
  ❖ Range-based: needs ordinal partitioning attribute(s)
❖ Tradeoffs: Round-robin often inefficient for parallel query processing (why?); range-based good for range queries but faces new kind of “skew”; hashing-based is most common
❖ Replication often used for more availability, performance
Parallel Scans and Select

- **Intra-operator parallelism** is our primary focus
  - Inter-operator and inter-query parallelism also possible!
- **Filescan:**
  - Trivial! Worker simply scans its partition and streams it
  - Apply selection predicate (if any)
- **Indexed:**
  - Depends on data partitioning scheme and predicate!
  - Same tradeoffs: Hash index vs B+ Tree index
  - Each worker can have its own (sub-)index
  - Master *routes* query based on “matching workers”
Parallel Sorting

- **Naive algorithm**: (1) Each worker sorts local partition (EMS); (2) Master merges all locally sorted runs
  - **Issue**: Parallelism is limited during merging phase!

- **Faster algorithm**: (1) Scan in parallel and range partition data (most likely a repartitioning) based on SortKey; (2) Each worker sorts local allotted range (EMS); result is globally sorted and conveniently range-partitioned
  - **Potential Issue**: Skew in range partitions; handled by roughly estimating distribution using sampling
Parallel Sorting

Original Partitions

Worker 1

$V_1$ to $V_2$

Assign SortKey

Worker 2

$V_2$ to $V_3$

Master

Worker $n$

$V_{n-1}$ to $V_n$

Range-partitioned

Worker 1

$V_1$ to $V_2$

Range splits

Worker 2

$V_2$ to $V_3$

Worker $n$

$V_{n-1}$ to $V_n$

Re-partitioning

Local EMS

Worker 1

$V_1$ to $V_2$

Worker 2

$V_2$ to $V_3$

Worker $n$

$V_{n-1}$ to $V_n$

Globally Sorted

Worker 1

$V_1$ to $V_2$

Master

Worker 2

$V_2$ to $V_3$

Worker $n$

$V_{n-1}$ to $V_n$
Parallel Aggregates and Group By

- **Without Group By List:**
  - Trivial for MAX, MIN, COUNT, SUM, AVG (why?)
  - MEDIAN requires parallel sorting (why?)

- **With Group By List:**
  1. If AggFunc allows, pre-compute partial aggregates
  2. Master assigns each worker a set of groups (hash partition)
  3. Each worker communicates its partial aggregate for a group to that group’s assigned worker (aka “shuffle”)
  4. Each worker finishes aggregating for all its assigned groups
Parallel Group By Aggregate

Original Partitions

Worker 1

Worker 2

Worker n

Local GrpBY

Partial Aggs

Worker 1

Worker 2

Worker n

Assign GroupingList

Hash splits

Master

Re-partitioned Partial Aggs

Worker 1

Worker 2

Worker n

Local GrpBY Again

Final Aggs

Worker 1

Worker 2

Worker n

Local

GrpBY

Again

Final

Aggs
Parallel Project

- **Non-deduplicating Project:**
  - Trivial! Pipelined with Scans/Select

- **Deduplicating Project:**
  - Each worker deduplicates its partition on ProjectionList
  - If estimated output size is small (catalog?), workers communicate their results to master to finish dedup.
  - If estimated output size is too large for master’s disk, similar algorithm as Parallel Aggregate with Group By, except, there is no AggFunc computation
Parallel Nested Loops Join

- Given two tables A and B and JoinAttribute for equi-join
  1. Master assigns range/hash splits on JoinAttribute to workers
  2. Repartitioning of A and B separately using same splits on JoinAttribute (unless pre-partitioned on it!)
  3. Worker i applies BNLJ locally on its partitions Ai and Bi
  4. Overall join output is just collection of all n worker outputs

- If join is not equi-join, there might be a lot of communication between workers; worst-case: all-to-all for cross-product!
Parallel “Split” and “Merge” for Joins

❖ Repartitioning quite common for parallel (equi-)joins
❖ Functionality abstracted as two new physical operators:
  ❖ **Split**: each worker sends a subset of its partition to another worker based on master’s command (hash/range)
  ❖ **Merge**: each worker unions subsets sent to it by others and constructs its assigned (re)partitioned subset
❖ Useful for parallel BNLJ, Sort-Merge Join, and Hash Join
Parallel Sort-Merge and Hash Join

- For SMJ, split is on ranges of (ordinal) JoinAttribute; for HJ, split is on hash function over JoinAttribute
- Worker i does local join of Ai and Bi using SMJ or HJ
Improved Parallel Hash Join

- 2-phase parallel HJ to improve performance
- **Idea:** Previous version hash partitions JoinAttribute to n (same as # workers); instead, decouple the two and do a 2-stage process: partition phase and join phase
- **Partition Phase:** Say $|A| < |B|; divide A and B into k (can be > n) partitions using $h1()$ s.t. each $F \times |A_i| < $ Cluster RAM
- **Join Phase:** Repartition an $A_i$ into n partitions using $h2()$; build hash table on new $A_{ij}$ at worker $j$ as tuples arrive; repartition $B_i$ using $h2()$; local HJ of $A_{ij}$ and $B_{ij}$ on worker $j$ in parallel for $j = 1$ to n; repeat all these steps for each $i = 1$ to k
- Uses all n workers for join of each subset pair $A_i \bowtie B_i$
Parallel Query Optimization

- Far more complex than single-node QO!
- I/O cost, CPU cost, and communication cost for each phy. op.
- Space of PQPs explodes: each node can have its own different local sub-plan (e.g., filescan v indexed)
- Pipeline parallelism and partitioned parallelism can be interleaved in complex ways!
- Join order enumeration affected: bushy trees can be good!
- … (we will skip more details)
Parallel vs “Distributed” RDBMSs

- A parallel RDBMS layers distribution atop the file system
  - Can handle dozens of nodes (Gamma, Teradata, etc.)
- Raghu’s “distributed”: collection of “independent” DBMSs
  - Quirk of terminology; “federated” more accurate term
  - Each base RDBMS can be at a different location!
  - Each RDBMS might host a subset of the database files
  - Might need to ship entire files for distributed QP
- … (we will skip more details)
- These days: “Polystores,” federated DBMSs on steroids!
Cloud Computing

- Compute, storage, memory, networking are virtualized and exist on remote servers; rented by application users
  - **Manageability**: Managing hardware is not user's problem!
  - **Pay-as-you-go**: Fine-grained pricing economics based on actual usage (granularity: seconds to years!)
  - **Elasticity**: Can dynamically add or reduce capacity based on actual workload’s demand
- Infrastructure-as-a-Service (IaaS); Platform-as-a-Service (PaaS); Software-as-a-Service (SaaS)
How to redesign a parallel RDBMS to best exploit the cloud’s capabilities?
Evolution of Cloud Infrastructure

❖ **Data Center**: Physical space from which a cloud is operated

❖ **3 generations of data centers/clouds:**

❖ **Cloud 1.0 (Past)**: Networked servers; user rents/time-sliced access to servers needed for data/software

❖ **Cloud 2.0 (Current)**: “Virtualization” of networked servers; user rents amount of resource capacity; cloud provider has a lot more flexibility on provisioning (multi-tenancy, load balancing, more elasticity, etc.)

❖ **Cloud 3.0 (Ongoing Research)**: “Serverless” and disaggregated resources all connected to fast networks
Revisiting Parallelism in the Cloud

Networks have become much faster: 100GbE to even TbE!

Such bundling could under-utilize resources!

How to exploit cloud’s virtualization of compute, memory, and storage resources to improve speed and utilization?
Cloud-Native Parallel RDBMSs

❖ Not just running a regular parallel RDBMS on IaaS!
❖ Need to revisit, redesign, and reimplemented storage subsystem, memory management, query processing and optimization, transaction management, and more!
❖ Higher levels (data model, SQL, parser, etc.) preserved
❖ Cloud providers, traditional database companies, startups
Key Example: SnowFlake
Key Example: AWS Redshift
Outline

❖ Parallel RDBMSs and Cloud-Native RDBMSs
❖ Beyond RDBMSs: A Brief History
❖ “Big Data” Systems
Beyond RDBMSs: A Brief History

- Relational model and RDBMSs are too restrictive:
  1. “Flat” tables with few data/attribute types
  2. Restricted language interface (SQL)
  3. Need to know schema first!
  4. Optimized for static dataset

**Object-Relational DBMSs**: UDT, UDFs, text, multimedia, etc.

- **PL/SQL**: recursive SQL; embedded SQL; QBE; visual interfaces
- **“Schema-later” semi-structured XML data model; XQuery**
- **Stream data model**: “standing” queries; time windows

Advertisement: Take CSE 135 and CSE 231B to learn more!
So, why did people still need to look beyond RDBMSs?
The DB community got blindsided by the unstoppable rise of the Web/Internet giants!

DB folks underappreciated 4 key concerns of Web folks:

- Developability
- Fault Tolerance
- Elasticity
- Cost/Politics!
DB/Enterprise vs. Web Dichotomy

- DB folks underappreciated 4 key concerns of Web folks:
  - **Developability**: RDBMS extensibility mechanisms (UDTs, UDFs, etc.) are too painful to use for programmers!

  **DB companies**: we write the software and sell to our customers, viz., *enterprise* companies (banks, retail, etc.)

  **Web companies**: we will hire an army of software engineers to build own in-house software systems!

  *Need simpler APIs and DBMSs that scale custom programs*
DB folks underappreciated 4 key concerns of Web folks:

**Fault Tolerance**: What if we run on 100Ks of machines?!

**DB companies**: our customers do not need more than a few dozen machines to store and analyze their data!

**Web companies**: we need hundreds of thousands of machines for planetary-scale Web services!

*If a machine fails, user should not have to rerun entire query!*

*DBMS should take care of fault tolerance, not user/appl.*
DB folks underappreciated 4 key concerns of Web folks:

**Elasticity**: Resources should adapt to “query” workload

**DB companies**: our customers have “fairly predictably” sized datasets and workloads; can fix their clusters!

**Web companies**: our workloads could vary widely and the datasets they need vary widely!

*Need to be able to upsize and downsize clusters easily on-the-fly, based on current query workload*
DB/Enterprise vs. Web Dichotomy

❖ DB folks underappreciated 4 key concerns of Web folks:

**Cost/Politics**: Commercial RDBMS licenses too costly!

**DB companies**: our customers have $$$! 😊

**Web companies**: our products are mostly free (ads?); why pay so much $$$ if we can build our own DBMSs?

*Many started with MySQL (!) but then built their own DBMSs*

*New tools were free & open source; led to viral adoption!*
Cool, so, these new systems jolted the DB folks from being smug and complacent!
But what is “Big Data”?
“Big Data”

- Marketing term; think “Big” as in “Big Oil”, not “big building”
- Wikipedia says: “Data that is so large and complex that existing toolkits [read RDBMSs!] are not adequate [hah!]”
- Typical characterization by 3 Vs:
  - **Volume**: larger-than-RAM; >= TBs, even Exabytes!
  - **Variety**: relations, webpages, docs, tweets, multimedia, etc.
  - **Velocity**: high generation rate, e.g., sensors, surveillance, etc.
Why “Big Data” now? 1. Applications

- New “data-driven mentality” in almost all applications:
  - **Web**: search, e-commerce, e-mails, social media
  - **Science**: satellite imagery, CERN’s LHC, document corpora
  - **Medicine**: pharmacogenomics, precision medicine
  - **Logistics**: sensors, GPS, “Internet of Things”
  - **Finance**: high-throughput trading, monitoring
  - **Humanities**: digitized books/literature, social media
  - **Governance**: e-voting, targeted campaigns, NSA 😊
  - …
Why “Big Data” now? 2. Storage

Global Information Storage Capacity in optimally compressed bytes

- Paper, film, audiotape and vinyl: 6%
- Analog videotapes (VHS, etc.): 94%
- Portable media, flash drives: 2%
- Portable hard disks: 2.4%
- CDs and minidisks: 6.8%
- Computer servers and mainframes: 8.9%
- Digital tape: 11.8%
- DVD/Blu-ray: 22.8%
- PC hard disks: 44.5%
- Others: <1% (incl. chip cards, memory cards, floppy disks, mobile phones, PDAs, cameras/camcorders, video games)

% digital:
1% 3% 25% 94%

Outline

- Parallel RDBMSs and Cloud-Native RDBMSs
- Beyond RDBMSs: A Brief History
- “Big Data” Systems
  - The MapReduce/Hadoop Craze
  - Spark and Other Dataflow Systems
  - Key-Value NoSQL Systems
  - Graph Processing Systems
  - Advanced Analytics/ML Systems
The MapReduce/Hadoop Craze

❖ Blame Google!
❖ “Simple” problem: index, store, and search the Web! 😊
❖ Who were their major systems hires?
   Jeff Dean and Sanjay Ghemawat (Systems, not DB or IR)
❖ Why did they not use RDBMSs? (Haha.)
   Developability, data model, fault tolerance, scale, cost, …
   Engineers started with MySQL; abandoned it!
What is MapReduce?


- Programming model for writing data-parallel programs + distributed system architecture for processing large data
- Map and Reduce are terms/ideas from functional PL
- Engineer only implements the “logic” of Map and Reduce
- Libraries in Java, C++, etc. handle orchestration of data distribution, parallelization, etc. “under the covers”

Was radically easier for engineers to write programs with!
What is MapReduce?

- **Standard example**: count word occurrences in a doc corpus
- **Input**: A set of text documents (say, webpages)
- **Output**: A dictionary of unique words and their counts

```
function map (String docname, String doctext) : 
  for each word w in doctext :
    emit (w, 1)

function reduce (String word, Iterator partialCounts) :
  sum = 0
  for each pc in partialCounts :
    sum += pc
  emit (word, sum)
```

Hmmm, sounds suspiciously familiar … 😏

Part of MapReduce API
How MapReduce Works

- Parallel flow of control and data upon running the MapReduce program:

The overall MapReduce word count process

Input          Splitting          Mapping          Shuffling          Reducing          Final result

Deer Bear River Car Car River Deer Car Bear

Deer Bear River

Car Car River

Deer Car Bear

Deer, 1 Bear, 1 River, 1

Car, 1 Car, 1 River, 1

Deer, 1 Car, 1 Bear, 1

Bear, 1 Bear, 1

Car, 1 Car, 1 Car, 1

Deer, 1 Deer, 1

River, 1 River, 1

Bear, 2 Bear, 1

Car, 3 Car, 1

Deer, 2 Deer, 1

River, 2 River, 1

Bear, 2 Car, 3 Deer, 2 River, 2

Each “Mapper” and “Reducer” is a separate process; parallel!
Fault tolerance achieved using data replication
Q: How would you do the word counting in RDBMS / in SQL?

❖ First step: Transform text docs into relations and load (how?)

Part of a stage called Extract-Transform-Load (ETL)

Suppose we pre-divide each document into words and have the schema: DocWords (DocName, Word)

❖ Second step: a single, simple SQL query!

```
SELECT   Word, COUNT(*)
FROM      DocWords
GROUP BY  Word
ORDER BY  Word
```

Parallelism, scaling, etc. done by RDBMS “under the covers”
What is Hadoop then?

❖ Open-source impl. Of Google’s ideas; includes MapReduce model and a distributed file system (HDFS)

❖ **Summary**: User writes logic of *Map* and *Reduce* functions in API; input splitting, data distribution, shuffling, fault tolerance, etc. all handled by the Hadoop library “under the covers”

❖ Exploded in popularity! 100s of papers, 10s of products!

_A real “revolution” in scalable data processing that took the DB community by surprise!_
A Spectacular “War of the Worlds”

MapReduce: A major step backwards

By David DeWitt on January 17, 2008 4:20 PM | Permalink | Comments (44) | TrackBacks (1)

[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebreaker]

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we’ll begin here with our thoughts on MapReduce [1]. The recent trade press has been filled with news of the revolution of so-called “cloud computing.” This paradigm entails harnessing large numbers of commodity computing nodes to solve a computing problem. In effect, this suggests constructing a data center by lining up a large number of “jelly beans” rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all — it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

No declarativity! Filescan-based! DeWitt’s work on parallel DBMSs! Cheap rip-off of RDBMSs!
“Young Turks” vs. “Old Guard”?

Swift and scathing rebuttal from MapReduce/Hadoop world!

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

DBMSs too high-level/hard to use for low-level text ETL
Meant for “offline” fault-tolerant workloads on cheap nodes
Google awarded a patent for MapReduce (ahem)!
MapReduce/Hadoop not meant to be an RDBMS replacement
Enter Hybrid Systems!

- **Clever DB researches**: “Let’s get the best of both worlds!”
- Numerous projects on hybrid systems in industry/academia:

  **Programming model-level**: Bring declarativity from RDBMS world to MapReduce/Hadoop world

  - SQL dialect over Hadoop
  - Dataflow language over Hadoop

  **Systems-level**: Intermix system implementation ideas

  - HadoopDB from Yale U.
  - Microsoft Polybase
“Big Data” Systems

❖ Parallel RDBMSs and Cloud-Native RDBMSs
❖ Beyond RDBMSs: A Brief History
❖ “Big Data” Systems
  ❖ The MapReduce/Hadoop Craze
  ❖ Spark and Other Dataflow Systems
  ❖ Key-Value NoSQL Systems
  ❖ Graph Processing Systems
  ❖ Advanced Analytics/ML Systems
Extended dataflow programming model (subsumes most of RA; MapReduce); system (re)designed from ground up

**Agenda**: Unified system to handle relations, text, etc.; support more general distributed data processing

Tons of sponsors, gazillion bucks, unbelievable hype!

**Key idea**: exploit distributed memory to cache data

**Key novelty**: lineage-based fault tolerance, not replication

Open-sourced to Apache; commercialized as Databricks
What does Spark have?

**Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In NSDI 2012**

<table>
<thead>
<tr>
<th>Transformations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>map(f : T ⇒ U)</td>
<td>RDD[T] ⇒ RDD[U]</td>
</tr>
<tr>
<td>filter(f : T ⇒ Bool)</td>
<td>RDD[T] ⇒ RDD[T]</td>
</tr>
<tr>
<td>flatMap(f : T ⇒ Seq[U])</td>
<td>RDD[T] ⇒ RDD[U]</td>
</tr>
<tr>
<td>sample(fraction : Float)</td>
<td>RDD[T] ⇒ RDD[T] (Deterministic sampling)</td>
</tr>
<tr>
<td>groupByKey()</td>
<td>RDD[(K, V)] ⇒ RDD[(K, Seq[V])]</td>
</tr>
<tr>
<td>reduceByKey(f : (V, V) ⇒ V)</td>
<td>RDD[(K, V)] ⇒ RDD[(K, V)]</td>
</tr>
<tr>
<td>union()</td>
<td>(RDD[T], RDD[T]) ⇒ RDD[T]</td>
</tr>
<tr>
<td>join()</td>
<td>(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]</td>
</tr>
<tr>
<td>cogroup()</td>
<td>(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]</td>
</tr>
<tr>
<td>crossProduct()</td>
<td>(RDD[T], RDD[U]) ⇒ RDD[(T, U)]</td>
</tr>
<tr>
<td>mapValues(f : V ⇒ W)</td>
<td>RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)</td>
</tr>
<tr>
<td>sort(c : Comparator[K])</td>
<td>RDD[(K, V)] ⇒ RDD[(K, V)]</td>
</tr>
<tr>
<td>partitionBy(p : Partitioner[K])</td>
<td>RDD[(K, V)] ⇒ RDD[(K, V)]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count()</td>
<td>RDD[T] ⇒ Long</td>
</tr>
<tr>
<td>collect()</td>
<td>RDD[T] ⇒ Seq[T]</td>
</tr>
<tr>
<td>reduce(f : (T, T) ⇒ T)</td>
<td>RDD[T] ⇒ T</td>
</tr>
<tr>
<td>lookup(k : K)</td>
<td>RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)</td>
</tr>
<tr>
<td>save(path : String)</td>
<td>Outputs RDD to a storage system, e.g., HDFS</td>
</tr>
</tbody>
</table>
Word Count Example in Spark

Spark has libraries for Python, Scala, and Java

```scala
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

```python
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" "))
  .map(lambda word: (word, 1))
  .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

```java
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
  .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
  .mapToPair(word -> new Tuple2<>(word, 1))
  .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```

SparkSQL offers an SQL-like front-end
Spark-based Ecosystem of Tools

The Berkeley Data Analytics Stack (BDAS)
How does Spark work?


```python
def add_demographics(events):
    u = sqlCtx.table("users")
    events \n        .join(u, events.user_id == u.user_id) \n        .withColumn("city", zipToCity(df.zip))
    events = add_demographics(sqlCtx.load("/data/events", "parquet"))
    training_data = events.where(events.city == "New York") .select(events.timestamp).collect()
```

Databricks is basically building yet another parallel RDBMS! 😊

Logical Plan

- filter
- join

Physical Plan

- join
- scan (events)
- filter

Physical Plan with Predicate Pushdown and Column Pruning

- join
- optimized scan (events)
- optimized scan (users)
Reinventing the Wheel?
Other Dataflow Systems

❖ Stratosphere/Apache Flink from TU Berlin
❖ Myria from U Washington
❖ AsterixDB from UC Irvine
❖ Azure Data Lakes from Microsoft
❖ …

Building such “Big Data” systems is (was?) one of the hottest topics in both industry and academia

My bias: Lot of system building; not sure of “research novelty”
References and More Material

❖ MapReduce/Hadoop:

❖ Spark:
❖ Online Tutorial: http://bit.ly/2r8lW0S
Outline

❖ Parallel RDBMSs and Cloud-Native RDBMSs
❖ Beyond RDBMSs: A Brief History
❖ “Big Data” Systems
❖ The MapReduce/Hadoop Craze
❖ Spark and Other Dataflow Systems
❖ Key-Value NoSQL Systems
❖ Graph Processing Systems
❖ Advanced Analytics/ML Systems
Key-Value NoSQL Systems

❖ **Simple API**: *get* and *put* unique records very quickly!
  ❖ Records usually uniquely identified by a “key”; information in record is the “value” (could be general JSON object)
❖ Used extensively by Web companies, e.g., get product record quickly and update stock count, update Facebook status, etc.
❖ Need high availability, high scalability, “eventual” consistency
❖ **Idea**: Discard ACID and 30+ years of DB lessons; use “BASE” (Basically Available, Soft state, and Eventually consistent)
❖ The new RDBMS-hating “movement” was christened “NoSQL”
Key-Value NoSQL Systems

Also called *transactional* NoSQL (read-write)

Hadoop / Spark aka *analytical* NoSQL (read mostly)
Key-Value NoSQL Systems

- Recent work on relaxed consistency models with guarantees in between full ACID and fuzzy best-effort BASE/Eventual

5 consistency levels of Microsoft Azure CosmosDB (a geo-distributed cloud-native DBMS)

My bias: Key area of research at the intersection of DB & distributed systems!

Advertisement: Take CSE 223B to learn more!
Outline

- Parallel RDBMSs and Cloud-Native RDBMSs
- Beyond RDBMSs: A Brief History
- “Big Data” Systems
  - The MapReduce/Hadoop Craze
  - Spark and Other Dataflow Systems
  - Key-Value NoSQL Systems
  - Graph Processing Systems
- Advanced Analytics/ML Systems
Graph Processing Systems

- Not a workload DB folks used to care much about
- Specialized graph systems have been around for years (Neo4j), but more popular now (Facebook, LinkedIn, etc.)
- **Data Model**: set of nodes, and set of (multi-)edges
- **Ops/queries**: nearest neighbors, shortest path, connectivity, density, cliques, etc.
Graph Processing Systems

Can be handled as an application on an RDBMS, but might be inefficient – transitive closure, repeated self-joins, etc.
Graph Processing Systems

Graphic showing the graph database space with various tools and systems like Microsoft, OrientDB, Neo4j, InfiniteGraph, Titan, Affinity, dx, and a graph representing Twitter and AllegroGraph.

Advertisement: “Graph Analytics” course on Coursera by UCSD