Programming at Scale: Dataflow and Consistency

cs378h
Questions?

Administrivia
• Rust lab due today!
• Project Proposal Due Thursday!

Agenda:
• Dataflow Wrapup
• Concurrency & Consistency at Scale
public void kmeans() {
    while(...) {
        for each point
            find_nearest_center(point);
        for each center
            compute_new_center(center)
    }
}
Review: K-Means

```java
public void kmeans() {
    while(...) {
        for each point in map {
            find_nearest_center(point);
        }
        for each center {
            compute_new_center(center);
        }
    }
}
```
Review: K-Means

```java
public void kmeans() {
    while(...) {
        for each point
            find_nearest_center(point);
        for each center
            compute_new_center(center)
    }
}
```
Review: K-Means

```java
public void kmeans() {
    while(...) {
        for each point
            find_nearest_center(point);
        for each center
            compute_new_center(center)
    }
}
```
Review: K-Means

public void kmeans() {
    while(...) {
        for each point
            find_nearest_center(point);
        for each center
            compute_new_center(center)
    }
}
Review: K-Means

```java
public void kmeans() {
    while(...) {
        for each point
            find_nearest_center(point);
        for each center
            compute_new_center(center);
    }
}
```
public void kmeans() {
    while(...) {
        map
        for each point
        find_nearest_center(point);
        reduce
        for each center
        compute_new_center(center)
    }
}
Review: K-Means

```java
public void kmeans() {
    while(...) {
        for each point
            find_nearest_center(point);
        for each center
            compute_new_center(center)
    }
}
```
Review: K-Means

public void kmeans() {
    while(...) {
        for each point
            find_nearest_center(point);
        for each center
            compute_new_center(center);
    }
}

/*
   * Map: find minimum distance center for point, emit to reducer
   * @Override
   *
   * public void map(LongWritable key, Text value,
                   * OutputCollector<DoubleWritable, DoubleWritable> output,
                   * Reporter reporter) throws IOException {
       * String line = value.toString();
       * double point = Double.parseDouble(line);
       * double min1, min2 = Double.MAX_VALUE, nearest_center = mCenters.get(0);
       * // Find the minimum distance from a point
       * for (Double c : mCenters) {
       *     min1 = c - point;
       *     if (Math.abs(min1) < Math.abs(min2)) {
       *         nearest_center = c;
       *         min2 = min1;
       *     }
       * }
       * // Emit the nearest center and the distance
       * output.collect(new DoubleWritable(nearest_center),
                      new DoubleWritable(min1));
   * */

   /*
   * Reduce: collect all points per center and calculate
   * the next center for those points
   * @Override
   *
   * public void reduce(
       *   DoubleWritable key, Iterator<DoubleWritable> values,
       *   OutputCollector<DoubleWritable, Text> output, Reporter reporter)
       * throws IOException {
       *   double newCenter;
       *   double sum = 0;
       *   int no_elements = 0;
       *   String points = "";
       *   while (values.hasNext()) {
       *     double d = values.next().get();
       *     points = points + " " + Double.toString(d);
       *     sum = sum + d;
       *   ++no_elements;
       * }
       * // We have a new center now
       * newCenter = sum / no_elements;
       * // Emit new center and point
       * output.collect(new DoubleWritable(newCenter), new Text(points));
   * */
Review: K-Means

```java
public void kmeans() {
    while(...) {
        for each point
            find_nearest_center(point);
        for each center
            compute_new_center(center);
    }
}
```

Key idea: **adapt workload to parallel patterns**

Questions:
- What kinds of computations can this express?
- What other patterns could we use?
How Does Parallelization Work?
Execution
Execution

Key idea $\rightarrow$ shuffle == sort!
Task Granularity And Pipelining

map tasks >> machines -- why?
Task Granularity And Pipelining

|map tasks| >> |machines| -- why?
Minimize fault recovery time
Pipeline map with other tasks
Easier to load balance dynamically
MapReduce: A major step backwards | The Database Column


on Jan 17 in Database architecture, Database history, Database innovation posted by DeWitt

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

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we’ll begin here with our views on MapReduce. This is a good time to discuss it, since the recent trade press has been filled with news of the revolution of so-called “cloud computing.” This paradigm entails harnessing large numbers of (low-end) processors working in parallel 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
MapReduce: not without Controversy

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
MapReduce: not without Controversy

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
MapReduce: not without Controversy

Backwards step in programming paradigm
MapReduce: not without Controversy

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
It’s just a group-by aggregate engine
MapReduce: not without Controversy

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
It’s just a group-by aggregate engine
Missing most DBMS features
  Schema, foreign keys, ...

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
MapReduce: not without Controversy

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
It’s just a group-by aggregate engine
Missing most DBMS features
Schema, foreign keys, ...
Incompatible with most DBMS tools

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
MapReduce: not without Controversy

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
  It’s just a group-by aggregate engine
Missing most DBMS features
  Schema, foreign keys, ...
Incompatible with most DBMS tools

So why is it such a big success?

1. A giant step backward in the programming paradigm for large-scale data intensive applications
Why is MapReduce backwards?
Why is MapReduce backwards?

Backwards step in programming paradigm
Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
   It’s just a group-by aggregate engine
Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
   It’s just a group-by aggregate engine
Missing most DBMS features
   Schema, foreign keys, ...
Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
   It’s just a group-by aggregate engine
Missing most DBMS features
   Schema, foreign keys, ...
Incompatible with most DBMS tools
Why is MapReduce backwards?

Backwards step in programming paradigm
Sub-optimal: brute force, no indexing
Not novel: 25 year-old ideas from DBMS lit
  It’s just a group-by aggregate engine
Missing most DBMS features
  Schema, foreign keys, …
Incompatible with most DBMS tools

So why is it such a big success?
MapReduce and Dataflow
MapReduce and Dataflow

- MR is a *dataflow* engine
MapReduce and Dataflow

- MR is a *dataflow* engine
MapReduce and Dataflow

- MR is a *dataflow* engine
- Lots of others
  - Dryad
  - DryadLINQ
  - Dandelion
  - CIEL
  - GraphChi/Pregel
  - Spark
MapReduce and Dataflow

- MR is a **dataflow** engine
- Lots of others
  - Dryad
  - DryadLINQ
  - Dandelion
  - CIEL
  - GraphChi/Pregel
  - Spark
MapReduce and Dataflow

- MR is a **dataflow** engine
- Lots of others
  - Dryad
  - DryadLINQ
  - Dandelion
  - CIEL

**Taxonomies:**
- DAG instead of BSP
- Interface variety
  - Memory FIFO
  - Disk
  - Network
- Flexible Modular Composition
Dryad (2007): 2-D Piping

- Unix Pipes: 1-D
  
grep | sed | sort | awk | perl
Dryad (2007): 2-D Piping

- Unix Pipes: 1-D
  \[\text{grep} \mid \text{sed} \mid \text{sort} \mid \text{awk} \mid \text{perl}\]

- Dryad: 2-D
  \[
  \text{grep}^{1000} \mid \text{sed}^{500} \mid \text{sort}^{1000} \mid \text{awk}^{500} \mid \text{perl}^{50}
  \]
Dataflow Engines
Dataflow Job Structure
Dataflow Job Structure

Input files

Channels

grep

sed

grep

Vertices (processes)

Stage

sort

sort

Output files

How to implement?

awk

perl
Channels

Finite streams of items

- distributed filesystem files (persistent)
- SMB/NTFS files (temporary)
- TCP pipes (inter-machine)
- memory FIFOs (intra-machine)
Channels

Finite streams of items

- distributed filesystem files (persistent)
- SMB/NTFS files (temporary)
- TCP pipes (inter-machine)
- memory FIFOs (intra-machine)

Key idea:
Encapsulate data movement behind channel abstraction \( \rightarrow \) gets programmer out of the picture
Commodity clusters: important platform

In **industry**: search, machine translation, ad targeting, ...
In **research**: bioinformatics, NLP, climate simulation, ...

Cluster-scale models (e.g. MR) de facto standard
Fault tolerance through replicated durable storage
Dataflow is the common theme
Commodity clusters: important platform

**In industry:** search, machine translation, ad targeting, ...

**In research:** bioinformatics, NLP, climate simulation, ...

Cluster-scale models (e.g. MR) de facto standard
Fault tolerance through replicated durable storage
Dataflow is the common theme

Multi-core
Iteration
Motivation

Programming models for clusters transform data flowing from stable storage to stable storage

E.g., MapReduce:
Motivation

Programming models for clusters transform data flowing from stable storage to stable storage

E.g., MapReduce:

**Benefits of data flow:** runtime can decide where to run tasks and can automatically recover from failures
Iterative Computations: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page’s rank to
   \[ \sum_{i \in \text{neighbors}} \frac{\text{rank}_i}{|\text{neighbors}_i|} \]

```scala
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
    ranks = links.join(ranks).flatMap {
        (url, (links, rank)) =>
        links.map(dest => (dest, rank/links.size))
    }.reduceByKey(_ + _)
}
```
Iterative Computations: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page’s rank to
   \[
   \sum_{i \in \text{neighbors}} \frac{\text{rank}_i}{|\text{neighbors}_i|}
   \]

   \[
   \text{links} = // \text{RDD of (url, neighbors) pairs}
   \text{ranks} = // \text{RDD of (url, rank) pairs}
   \]

   \[
   \text{for } (i <- 1 \text{ to ITERATIONS}) { }
   \text{ranks} = \text{links.join(ranks).flatMap} { }
   (url, (links, rank)) =>
   \text{links.map(dest => (dest, rank/links.size))}
   \text{.reduceByKey(_ + _)}
   \]
Iterative Computations: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page’s rank to
   \[ \Sigma_{i \in \text{neighbors}} \text{rank}_i / |\text{neighbors}_i| \]

\[
\begin{align*}
\text{links} &= /\text{ RDD of (url, neighbors) pairs} \\
\text{ranks} &= /\text{ RDD of (url, rank) pairs}
\end{align*}
\]

\[
\begin{align*}
\text{for (i <- 1 to ITERATIONS) } \{ \\
\text{ranks} &= \text{links}.join(\text{ranks}).flatMap \{ \\
&\quad \text{links}(\text{url}, (\text{links}, \text{rank})) \Rightarrow \\
&\quad \quad \text{links}.map(\text{dest} \Rightarrow (\text{dest}, \text{rank}/\text{links}.size)) \\
&\quad \} \text{reduceByKey}(+) \\
\}
\end{align*}
\]

**Solution:** augment data flow model with “resilient distributed datasets” (RDDs)
Programming Model

• Resilient distributed datasets (RDDs)
  • Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  • Created by transforming data in stable storage using data flow operators (map, filter, group-by, ...)
  • Can be cached across parallel operations
Programming Model

• Resilient distributed datasets (RDDs)
  • Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  • Created by transforming data in stable storage using data flow operators (map, filter, group-by, ...)
  • Can be cached across parallel operations

• Parallel operations on RDDs
  • Reduce, collect, count, save, ...
Programming Model

• Resilient distributed datasets (RDDs)
  • Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  • Created by transforming data in stable storage using data flow operators (map, filter, group-by, ...)
  • Can be cached across parallel operations

• Parallel operations on RDDs
  • Reduce, collect, count, save, ...

• Restricted shared variables
  • Accumulators, broadcast variables
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
```
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
```
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
```
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
```
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('t')(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('t')(2))
cachedMsgs = messages.cache()
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("	")(2))
cachedMsgs = messages.cache()
cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\t')(2))
cachedMsgs = messages.cache()
cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns.

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```python
cachedMsgs = messages.cache()
```

```scala
cachedMsgs.filter(_.contains("bar")).count
```

```java
cachedMsgs.filter(_.contains("foo")).count
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns

```python
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
```
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
...
Example: Log Mining

• Load error messages from a log into memory, then interactively search for various patterns

driver = spark.textFile("hdfs://...")
errors = driver.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count

Result: full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)
• RDDs maintain *lineage* information that can be used to reconstruct lost partitions

• Ex:

```scala
cachedMsgs = textFile(...).filter(_.contains("error")).map(_.split('t')(2)).persist()
```
Data-Parallel Computation Systems

- Application
- Language
- Execution
- Storage
Data-Parallel Computation Systems

Application

Language

Execution

Storage

Parallel Databases

SQL
Data-Parallel Computation Systems

Application

Language

Execution

Storage

SQL

Parallel Databases

Map-Reduce

GFS

BigTable
Data-Parallel Computation Systems

- Application
- Language
- Execution
- Storage

- SQL
- Sawzall

- Parallel Databases
  - Map-Reduce
  - GFS
  - BigTable
Data-Parallel Computation Systems

- Application
- Language
- Execution
- Storage

- SQL
- Sawzall

- Parallel Databases
- Map-Reduce
- GFS
- BigTable

```plaintext
count: table sum of int;
total: table sum of float;
sum_of_squares: table sum of float;
x: float = input;
emit count <- 1;
emit total <- x;
emit sum_of_squares <- x * x;
```
Data-Parallel Computation Systems

- Application
  - SQL
  - Sawzall

- Language
  - Sawzall
  - Map-Reduce

- Execution
  - Parallel Databases
  - GFS
  - BigTable

- Storage
Data-Parallel Computation Systems

- Application
- Language
- Execution
- Storage

Parallel Databases
- SQL
- Sawzall
- Map-Reduce
- Hadoop
- GFS
- BigTable
- HDFS
- S3
Data-Parallel Computation Systems

**Application**

**Language**

**Execution**

**Storage**

**Parallel Databases**

- Map-Reduce
  - GFS
  - BigTable

- ≈SQL
  - Pig, Hive

- Hadoop
  - HDFS
  - S3

- Sawzall
  - SQL
Data - Parallel Computation Systems

- Execution
- Application
- Storage
- Language

Parallel Databases

- SQL
- Sawzall
- ≈SQL
- Pig, Hive
- S3

- BigTable

---

```sql
lines = LOAD '/user/hadoop/HDFS_File.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped GENERATE group, COUNT(words);
DUMP wordcount;
```

---

```java
-- import the file as lines
CREATE EXTERNAL TABLE lines(line string)
LOAD DATA INPATH 'books' OVERWRITE INTO TABLE lines;

-- create a virtual view that splits the lines
SELECT word, count(*) FROM lines
    LATERAL VIEW explode(split(text, ' ')) lTable as word
GROUP BY word;
```
Data-Parallel Computation Systems

Application

Language

Execution

Storage

SQL

Sawzall

≈SQL

Pig, Hive

Map-Reduce

Hadoop

Parallel Databases

GFS

BigTable

HDFS

S3

Sawzall
Data-Parallel Computation Systems

Application

Language

Execution

Storage

Parallel Databases

SQL

Sawzall

≈SQL

Map-Reduce

Pig, Hive

DryadLINQ

Dryad

Hadoop

GFS

HDFS

BigTable

S3

Cosmos

Azure

SQL Server

≈SQL

Sawzall
Data-Parallel Computation Systems

Application

Language

Execution

Storage

Parallel Databases

Sawzall

Map-Reduce

GFS

BigTable

Hadoop

HDFS

S3

≈SQL

Pig, Hive

DryadLINQ

Scope

Dryad

Cosmos, HPC, Azure

Cosmos

Azure

SQL Server

SQL

≈SQL

LINQ, SQL

DryadLINQ

Scope
Data-Parallel Computation Systems

Application

Language

Execution

Storage

Parallel Databases

SQL

Sawzall

≈SQL

LINQ, SQL

Parallel

Databases

Map-Reduce

GFS

BigTable

Hadoop

HDFS

S3

DryadLINQ

Spark

HPC, Azure

Cosmos

Azure

SQL Server

Execution

Application

Storage

Language
(Yet) Another Framework
(Yet) Another Framework

Consistency
(Yet) Another Framework
(Yet) Another Framework
(Yet) Another Framework
(Yet) Another Framework

- **Strong: ACID**
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- **Eventual: BASE**

Consistency

Data Model

Implementation Techniques
(Yet) Another Framework

### Data Model
- Basically Available
- Soft State
- Eventually Consistent

### Implementation Techniques

#### Strong: ACID
- Atomicity
- Consistency
- Isolation
- Durability

#### Eventual: BASE

### Consistency
(Yet) Another Framework

Data Model

Consistency

Strong: ACID

Eventual: BASE

Implementation Techniques
(Yet) Another Framework

Consistency

Strong: ACID  Eventual: BASE

Key Value Stores

Data Model

Implementation Techniques
(Yet) Another Framework

Key Value Stores

Document Stores

Strong: ACID

Eventual: BASE

Consistency

Data Model

Implementation Techniques
(Yet) Another Framework

- Wide-Column Stores
- Key Value Stores
- Document Stores

**Consistency**
- Strong: ACID
- Eventual: BASE

Data Model

Implementation Techniques
(Yet) Another Framework

- **Key Value Stores**
- **Document Stores**
- **Wide-Column Stores**

**Consistency**
- Strong: ACID
- Eventual: BASE

**Sharding/Partitioning**

**Data Model**

**Implementation Techniques**
(Yet) Another Framework

Data Model

Consistency

Strong: ACID

Eventual: BASE

Sharding/Partitioning

Replication

Implementation Techniques
(Yet) Another Framework

Data Model

Consistency

Strong: ACID

Key Value Stores

Eventual: BASE

Wide-Column Stores

Document Stores

Sharding/Partitioning

Replication

Storage

Implementation Techniques
(Yet) Another Framework

Key Value Stores

Document Stores

Wide-Column Stores

Strong: ACID

Eventual: BASE

Consistency

Sharding/Partitioning

Replication

Storage

Query Support

Implementation Techniques

Data Model
(Yet) Another Framework

- Key Value Stores
- Document Stores
- Wide-Column Stores

**Consistency**
- Strong: ACID
- Eventual: BASE

**Implementation Techniques**
- Shared-Disk
- Range-Sharding
- Hash-Sharding
- Consistent Hashing

**Sharding/Partitioning**
- Replication
- Storage
- Query Support
(Yet) Another Framework

Data Model

- Key Value Stores
- Document Stores
- Wide-Column Stores

Consistency

- Strong: ACID
- Eventual: BASE

Implementation Techniques

- Sharding/Partitioning
- Replication
- Storage
- Query Support
(Yet) Another Framework

Data Model

- Key Value Stores
- Document Stores
- Wide-Column Stores

Consistency

- Strong: ACID
- Eventual: BASE

Implementation Techniques

- Primary-Backup
- Commit-Consensus Protocol
- Sync/Async
(Yet) Another Framework

Data Model

Consistency

Strong: ACID  Eventual: BASE

Key Value Stores

Document Stores

Wide-Column Stores

Sharding/Partitioning

Replication

Storage

Query Support

Implementation Techniques
(Yet) Another Framework

Key Value Stores

Document Stores

Wide-Column Stores

Strong: ACID

Eventual: BASE

Consistency

Sharding/Partitioning

Replication

Storage

Query Support

Implementation Techniques

• Logging
• Update In Place
• Caching
• In-Memory Storage
(Yet) Another Framework
(Yet) Another Framework

Wide-Column Stores

[Yet] Another Framework

Consistency

Key Value Stores

Data Model

Strong: ACID

Document Stores

Eventual: BASE

Wide-Column Stores

Sharding/Partitioning

Replication

Storage

Query Support

Implementation Techniques

• Secondary Indexing
• Query Planning
• Materialized Views
• Analytics
(Yet) Another Framework

- **Key Value Stores**
- **Document Stores**
- **Wide-Column Stores**

**Consistency**
- Strong: ACID
- Eventual: BASE

**Implementation Techniques**
- Sharding/Partitioning
- Replication
- Storage
- Query Support
(Yet) Another Framework

Still not a perfect framework

Cons:
- Many dimensions contain sub-dimensions
- Many concerns fundamentally coupled
- Dimensions are often un- or partially-ordered

Pros:
- Makes important concerns explicit
- Cleanly taxonomizes most modern systems
Consistency

<table>
<thead>
<tr>
<th>col</th>
<th>col</th>
<th>col₂</th>
<th>...</th>
<th>colₙ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consistency

How to keep data in sync?
### Consistency

How to keep data in sync?

- Partitioning → single row spread over multiple machines
Consistency

How to keep data in sync?

• Partitioning → single row spread over multiple machines
Consistency

How to keep data in sync?

• Partitioning → single row spread over multiple machines
Consistency

How to keep data in sync?

• Partitioning → single row spread over multiple machines
• Redundancy → single datum spread over multiple machines
Consistency

How to keep data in sync?

• Partitioning → single row spread over multiple machines
• Redundancy → single datum spread over multiple machines
Consistency

How to keep data in sync?

• Partitioning → single row spread over multiple machines
• Redundancy → single datum spread over multiple machines
Consistency: the core problem

writer \(\xrightarrow{\text{Write}(k,v)}\) R1 \(\xleftarrow{\text{Read}(k,v)}\) R2 \(\xrightarrow{}\) reader
Consistency: the core problem

- Clients perform reads and writes
Consistency: the core problem

- Clients perform reads and writes
- Data is replicated among a set of servers
Consistency: the core problem

- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
Consistency: the core problem

- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes
Consistency: the core problem

- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes

**How should we implement write?**
Consistency: the core problem

- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes

- How should we \textit{implement} write?
- How to \textit{implement} read?
Consistency: CAP Theorem
Consistency: CAP Theorem

- A distributed system can satisfy at most 2/3 guarantees of:
Consistency: CAP Theorem

• A distributed system can satisfy at most 2/3 guarantees of:
  1. Consistency:
Consistency: CAP Theorem

• A distributed system can satisfy at most 2/3 guarantees of:
  
  1. **Consistency:**
  
     • all nodes see same data at any time
     • or reads return latest written value by any client
Consistency: CAP Theorem

- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
Consistency: CAP Theorem

• A distributed system can satisfy at most 2/3 guarantees of:

  1. Consistency:
     • all nodes see same data at any time
     • or reads return latest written value by any client

  2. Availability:
     • system allows operations all the time,
     • and operations return quickly
Consistency: CAP Theorem

• A distributed system can satisfy at most 2/3 guarantees of:

1. **Consistency:**
   • all nodes see same data at any time
   • or reads return latest written value by any client

2. **Availability:**
   • system allows operations all the time,
   • and operations return quickly

3. **Partition-tolerance:**
Consistency: CAP Theorem

• A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     • all nodes see same data at any time
     • or reads return latest written value by any client
  2. **Availability**:
     • system allows operations all the time,
     • and operations return quickly
  3. **Partition-tolerance**:
     • system continues to work in spite of network partitions
Consistency: CAP Theorem

A distributed system can satisfy at most 2/3 guarantees of:

1. **Consistency**:
   - all nodes see same data at any time
   - or reads return latest written value by any client

2. **Availability**:
   - system allows operations all the time,
   - and operations return quickly

3. **Partition-tolerance**:
   - system continues to work in spite of network partitions
Consistency: CAP Theorem

• A distributed system can satisfy at most 2/3 guarantees of:

  1. Consistency:
     • all nodes see same data at any time
     • or reads return latest written value by any client

  2. Availability:
     • system allows operations all the time,
     • and operations return quickly

  3. Partition-tolerance:
     • system continues to work in spite of network partitions

Why care about CAP Properties?

Availability
• Reads/writes complete reliably and quickly.
• E.g. Amazon, each ms latency $\rightarrow$ $6M$ yearly loss.

Partitions
• Internet router outages
• Under-sea cables cut
• Rack switch outage
• system should continue functioning normally!

Consistency
• all nodes see same data at any time, or reads return latest written value by any client.
• This basically means correctness!
Consistency: CAP Theorem

- A distributed system can satisfy at most 2/3 guarantees of:

  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client

  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly

  3. **Partition-tolerance**:
     - system continues to work in spite of network partitions

Why care about CAP Properties?

**Availability**
- Reads/writes complete reliably and quickly.
- E.g. Amazon, each ms latency → $6M yearly loss.

**Partitions**
- Internet router outages
- Under-sea cables cut
- Rack switch outage
- system should continue functioning normally!

**Consistency**
- all nodes see same data at any time, or reads return latest written value by any client.
- This basically means correctness!
Consistency: CAP Theorem

• A distributed system can satisfy at most 2/3 guarantees of:

1. **Consistency**:  
   • all nodes see same data at any time  
   • or reads return latest written value by any client

2. **Availability**:  
   • system allows operations all the time,  
   • and operations return quickly

3. **Partition-tolerance**:  
   • system continues to work in spite of network partitions

Why is this “theorem” true?
Consistency: CAP Theorem

• A distributed system can satisfy at most 2/3 guarantees of:
  
  1. **Consistency:**
     • all nodes see same data at any time
     • or reads return latest written value by any client
  
  2. **Availability:**
     • system allows operations all the time,
     • and operations return quickly
  
  3. **Partition-tolerance:**
     • system continues to work in spite of network partitions

Why is this “theorem” true?
Consistency: CAP Theorem

• A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     • all nodes see same data at any time
     • or reads return latest written value by any client
  2. **Availability**:
     • system allows operations all the time,
     • and operations return quickly
  3. **Partition-tolerance**:
     • system continues to work in spite of network partitions

**Why is this “theorem” true?**
Consistency: CAP Theorem

- A distributed system can satisfy at most 2/3 guarantees of:
  1. **Consistency**:
     - all nodes see same data at any time
     - or reads return latest written value by any client
  2. **Availability**:
     - system allows operations all the time,
     - and operations return quickly
  3. **Partition-tolerance**:
     - system continues to work in spite of network partitions

Why is this “theorem” true?

- if(partition) { keep going } → !consistent && available
Consistency: CAP Theorem

• A distributed system can satisfy at most 2/3 guarantees of:

  1. **Consistency:**
     • all nodes see same data at any time
     • or reads return latest written value by any client

  2. **Availability:**
     • system allows operations all the time,
     • and operations return quickly

  3. **Partition-tolerance:**
     • system continues to work in spite of network partitions

**Why care about CAP Properties?**

**Availability**
• Reads/writes complete reliably and quickly.
  • E.g. Amazon, each ms latency → $6M yearly loss.

**Partitions**
• Internet router outages
• Under-sea cables cut
• Rack switch outage
  • system should continue functioning normally!

**Consistency**
• all nodes see same data at any time, or reads return latest written value by any client.
  • This basically means correctness!

**Why is this “theorem” true?**

if(partition) { keep going } → !consistent && available
if(partition) { stop } → consistent && !available
CAP Implications

- A distributed storage system can achieve at most two of C, A, and P.
- When partition-tolerance is important, you have to choose between consistency and availability.
CAP Implications

- A distributed storage system can achieve at most two of C, A, and P.
- When partition-tolerance is important, you have to choose between consistency and availability.

Consistency
- HBase, HyperTable, BigTable, Spanner
- RDBMSs (non-replicated)

Partition-tolerance
Availability
- Cassandra, RIAK, Dynamo, Voldemort

CAP is flawed
CAP Implications

- A distributed storage system can achieve at most two of C, A, and P.
- When partition-tolerance is important, you have to choose between consistency and availability.

PACELC:

```java
if(partition) {
    choose A or C
} else {
    choose latency or consistency
}
```

CAP is flawed
Consistency Spectrum

Faster reads and writes

More consistency

Eventual → Strong (e.g., Sequential)
Spectrum Ends: Eventual Consistency

- **Eventual Consistency**
  - If writes to a key stop, all replicas of key will converge
  - Originally from Amazon’s Dynamo and LinkedIn’s Voldemort systems

---

Faster reads and writes

---

More consistency

Strong (e.g., Sequential)
Spectrum Ends: Strong Consistency

• **Strict:**
  - Absolute time ordering of all shared accesses, reads always return last write

• **Linearizability:**
  - Each operation is visible (or available) to all other clients in real-time order

• **Sequential Consistency** [Lamport]:
  - "... the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.
  - After the fact, find a “reasonable” ordering of the operations (can re-order operations) that obeys sanity (consistency) at all clients, and across clients.

• **ACID** properties
Many *Many* Consistency Models

- Causal
- Red-Blue
- Probabilistic
- Eventual
- Per-key sequential
- CRDTs
- Strong (e.g., Sequential, Strict)
Many Many Consistency Models

- Amazon S3 – eventual consistency
- Amazon Simple DB – eventual or strong
- Google App Engine – strong or eventual
- Yahoo! PNUTS – eventual or strong
- Windows Azure Storage – strong (or eventual)
- Cassandra – eventual or strong (if R+W > N)
- ...

CRDTs

Causal

Red-Blue

Probabilistic

Eventual

Per-key sequential

Strong
(e.g., Sequential, Strict)
Question: How to choose what to use or support?

- Amazon S3 – eventual consistency
- Amazon Simple DB – eventual or strong
- Google App Engine – strong or eventual
- Yahoo! PNUTS – eventual or strong
- Windows Azure Storage – strong (or eventual)
- Cassandra – eventual or strong (if R+W > N)
- ...

Many Many Consistency Models

- Causal
- Red-Blue
- Probabilistic
- Eventual
- Per-key sequential
- CRDTs
- Strong (e.g., Sequential, Strict)
Some Consistency Guarantees

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
</tbody>
</table>
## Some Consistency Guarantees

<table>
<thead>
<tr>
<th>Consistency Guaranee</th>
<th>Description</th>
<th>Consistency</th>
<th>Performance</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
<td>A</td>
<td>D</td>
<td>F</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
<td>D</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
<td>C</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
<td>C</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>
Some Consistency Guarantees

- **Strong Consistency**: See all previous writes.
- **Eventual Consistency**: See subset of previous writes.
- **Consistent Prefix**: See initial sequence of writes.
- **Bounded Staleness**: See all “old” writes.
- **Monotonic Reads**: See increasing subset of writes.
- **Read My Writes**: See all writes performed by reader.

*metric* = set of allowable read results

**strength**
The Game of Soccer
The Game of Soccer
The Game of Soccer
The Game of Soccer
The Game of Soccer

for half = 1 .. 2 {

The Game of Soccer

for half = 1 .. 2 {
    while half not over {
}
for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            
        }
    }
}
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                
```
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read ("visitors");
            }
        }
    }
}
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read ("visitors");
                Write ("visitors", score + 1);
            }
        }
    }
}
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read ("visitors");
                Write ("visitors", score + 1);
            } else {
            }
        }
    }
}
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read ("visitors");
                Write ("visitors", score + 1);
            } else {
                score = Read ("home");
        } else {
            score = Read ("home");
        }
    }
} else {
    score = Read ("home");
}
for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read(“visitors”);
                Write(“visitors”, score + 1);
            } else {
                score = Read(“home”);
                Write(“home”, score + 1);
            }
        }
    }
}
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read ("visitors");
                Write ("visitors", score + 1);
            } else {
                score = Read ("home");
                Write ("home", score + 1);
            }
        }
    }
}
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read (“visitors”);
                Write (“visitors”, score + 1);
            } else {
                score = Read (“home”);
                Write (“home”, score + 1);
            }
        }
    }
}

hScore = Read (“home”);
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read ("visitors");
                Write ("visitors", score + 1);
            } else {
                score = Read ("home");
                Write ("home", score + 1);
            }
        }
    }
}

hScore = Read("home");
vScore = Read("visit");
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read ("visitors");
                Write ("visitors", score + 1);
            } else {
                score = Read ("home");
                Write ("home", score + 1);
            } }
    } }

hScore = Read("home");
vScore = Read("visit");
if (hScore == vScore)
for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read ("visitors");
                Write ("visitors", score + 1);
            } else {
                score = Read ("home");
                Write ("home", score + 1);
            }
        }
    }
    hScore = Read("home");
    vScore = Read("visit");
    if (hScore == vScore)
        play-overtime
The Game of Soccer

for half = 1 .. 2 {
    while half not over {
        kick-the-ball-at-the-goal
        for each goal {
            if visiting-team-scored {
                score = Read ("visitors");
                Write ("visitors", score + 1);
            } else {
                score = Read ("home");
                Write ("home", score + 1);
            }
        }
    }
}

hScore = Read("home");
vScore = Read("visit");
if (hScore == vScore)
    play-overtime
Official Scorekeeper

Let’s say we have a game where we want to keep track of the score. We can write a script to update the score whenever a visitor comes to the game.

```plaintext
score = Read ("visitors");
Write ("visitors", score + 1);
```

### Consistency Types

- **Strong Consistency**: See all previous writes.
- **Eventual Consistency**: See subset of previous writes.
- **Consistent Prefix**: See initial sequence of writes.
- **Monotonic Reads**: See increasing subset of writes.
- **Read My Writes**: See all writes performed by reader.
- **Bounded Staleness**: See all “old” writes.
Official Scorekeeper

score = **Read** (“visitors”);
**Write** (“visitors”, score + 1);

Desired consistency?

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
</tbody>
</table>
Official Scorekeeper

score = **Read** (“visitors”);
**Write** (“visitors”, score + 1);

Desired consistency?
**Strong**

<table>
<thead>
<tr>
<th>Strong Consistency</th>
<th>See all previous writes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
</tbody>
</table>
Official Scorekeeper

score = Read (“visitors”);
Write (“visitors”, score + 1);

Desired consistency?

Strong
= Read My Writes!

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
</tbody>
</table>
Official Scorekeeper

score = \textbf{Read} ("visitors");
\textbf{Write} ("visitors", score + 1);

Desired consistency?

\textbf{Strong} = \textbf{Read My Writes}!

<table>
<thead>
<tr>
<th>Strong Consistency</th>
<th>See all previous writes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all &quot;old&quot; writes.</td>
</tr>
</tbody>
</table>
Referee

vScore = \textbf{Read} ("visitors");
hScore = \textbf{Read} ("home");
if vScore == hScore
    play-overtime
vScore = Read (“visitors”);
hScore = Read (“home”);
if vScore == hScore
  play-overtime

Desired consistency?

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
</tbody>
</table>
Referee

vScore = Read ("visitors");
hScore = Read ("home");
if vScore == hScore
    play-overtime

Desired consistency?

Strong consistency

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all &quot;old&quot; writes.</td>
</tr>
</tbody>
</table>
Radio Reporter

do {
    BeginTx();
    vScore = Read (“visitors”);
    hScore = Read (“home”);
    EndTx();
    report vScore and hScore;
    sleep (30 minutes);
}
Radio Reporter

do {
    \textit{BeginTx();}
    \texttt{vScore = \textbf{Read} (“visitors”);}
    \texttt{hScore = \textbf{Read} (“home”);}
    \textit{EndTx();}
    \texttt{report vScore and hScore;}
    \texttt{sleep (30 minutes);} 
} 

Desired consistency?

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
</tbody>
</table>
Radio Reporter

```
do {
    BeginTx();
    vScore = Read ("visitors");
    hScore = Read ("home");
    EndTx();
    report vScore and hScore;
    sleep (30 minutes);
}
```

Desired consistency?

**Consistent Prefix**

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all &quot;old&quot; writes.</td>
</tr>
</tbody>
</table>
Radio Reporter

\[
do \{
    \text{BeginTx();}
    \text{vScore} = \text{Read ("visitors");}
    \text{hScore} = \text{Read ("home");}
    \text{EndTx();}
    \text{report vScore and hScore;}
    \text{sleep (30 minutes);}
\}
\]

Desired consistency?

- Consistent Prefix
- Monotonic Reads

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
</tbody>
</table>
Radio Reporter

```
do {
    BeginTx();
    vScore = Read ("visitors");
    hScore = Read ("home");
    EndTx();
    report vScore and hScore;
    sleep (30 minutes);
}
```

Desired consistency?
- Consistent Prefix
- Monotonic Reads
- or Bounded Staleness

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all &quot;old&quot; writes.</td>
</tr>
</tbody>
</table>
Radio Reporter

do {
    \textit{BeginTx();}
    vScore = \textbf{Read} ("visitors");
    hScore = \textbf{Read} ("home");
    \textit{EndTx();}
    report vScore and hScore;
    sleep (30 minutes);
}

Desired consistency?

\textbf{Consistent Prefix}

\textbf{Monotonic Reads}

or \textbf{Bounded Staleness}

\begin{tabular}{|l|l|}
\hline
\textbf{Strong Consistency} & See all previous writes. \\
\textbf{Eventual Consistency} & See subset of previous writes. \\
\textbf{Consistent Prefix} & See initial sequence of writes. \\
\textbf{Monotonic Reads} & See increasing subset of writes. \\
\textbf{Read My Writes} & See all writes performed by reader. \\
\textbf{Bounded Staleness} & See all "old" writes. \\
\hline
\end{tabular}
While not end of game {
    drink beer;
    smoke cigar;
}
go out to dinner;
vScore = \textbf{Read} (“visitors”);
hScore = \textbf{Read} (“home”);
write article;
Sportswriter

While not end of game {
  drink beer;
  smoke cigar;
}
go out to dinner;
vScore = Read ("visitors");
hScore = Read ("home");
write article;

Desired consistency?

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all &quot;old&quot; writes.</td>
</tr>
</tbody>
</table>
While not end of game {
  drink beer;
  smoke cigar;
}
go out to dinner;
vScore = \textbf{Read} ("visitors");
hScore = \textbf{Read} ("home");write article;

Desired consistency?

**Eventual**
Sportswriter

While not end of game {
    drink beer;
    smoke cigar;
}
go out to dinner;
vScore = Read ("visitors");
hScore = Read ("home");
write article;

Desired consistency?
Eventual
Bounded Staleness

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all &quot;old&quot; writes.</td>
</tr>
</tbody>
</table>
Statistician

Wait for end of game;
score = \textbf{Read} (“home”);
stat = \textbf{Read} (“season-goals”);
\textbf{Write} (“season-goals”, stat + score);

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Corresponding Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
</tbody>
</table>
Statistician

Wait for end of game;
score = Read ("home");
stat = Read ("season-goals");
Write ("season-goals", stat + score);

Desired consistency?

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
</tbody>
</table>
Wait for end of game; 
score = \textbf{Read} ("home"); 
stat = \textbf{Read} ("season-goals"); 
\textbf{Write} ("season-goals", stat + score);

Desired consistency? 
\textbf{Strong Consistency} (1st read)

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all &quot;old&quot; writes.</td>
</tr>
</tbody>
</table>
Statistician

Wait for end of game; 
score = Read ("home"); 
stat = Read ("season-goals"); 
Write ("season-goals", stat + score);

Desired consistency? 
**Strong Consistency** (1st read) 
**Read My Writes** (2nd read)

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all “old” writes.</td>
</tr>
</tbody>
</table>
Stat Watcher

do {
    stat = **Read** ("season-goals");
    discuss stats with friends;
    sleep (1 day);
}
Stat Watcher

do {
    stat = \textbf{Read} ("season-goals");
    discuss stats with friends;
    sleep (1 day);
} 

Desired consistency?

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all &quot;old&quot; writes.</td>
</tr>
</tbody>
</table>
Stat Watcher

do {
    stat = Read ("season-goals");
    discuss stats with friends;
    sleep (1 day);
}

Desired consistency?

**Eventual Consistency**

<table>
<thead>
<tr>
<th>Consistency Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Consistency</td>
<td>See all previous writes.</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td>See subset of previous writes.</td>
</tr>
<tr>
<td>Consistent Prefix</td>
<td>See initial sequence of writes.</td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>See increasing subset of writes.</td>
</tr>
<tr>
<td>Read My Writes</td>
<td>See all writes performed by reader.</td>
</tr>
<tr>
<td>Bounded Staleness</td>
<td>See all &quot;old&quot; writes.</td>
</tr>
</tbody>
</table>
Official scorekeeper:

score = \textbf{Read} ("visitors");
\textbf{Write} ("visitors", score + 1);

Referee:

Radio reporter:

do {
    vScore = \textbf{Read} ("visitors");
    hScore = \textbf{Read} ("home");
    report vScore and hScore;
    sleep (30 minutes);
}

Sportswriter:

While not end of game {
    drink beer;
    smoke cigar;
}
go out to dinner;
vScore = \textbf{Read} ("visitors");
hScore = \textbf{Read} ("home");
\textbf{Write} ("season-goals", stat + score);

Statistician:

wait for end of game;

stat = \textbf{Read} ("season-goals");
\textbf{Write} ("season-goals", stat + score);

discuss stats with friends;

Stat watcher:

stat = \textbf{Read} ("season-runs");
discuss stats with friends;
Sequential Consistency

- weaker than strict/strong consistency
  - All operations are executed in *some* sequential order
  - each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each process preserves its program order

<table>
<thead>
<tr>
<th>P1: W(x)a</th>
<th>P1: W(x)a</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2: W(x)b</td>
<td>P2: W(x)b</td>
</tr>
<tr>
<td>P3: R(x)b</td>
<td>P3: R(x)b</td>
</tr>
<tr>
<td>R(x)a</td>
<td>R(x)a</td>
</tr>
<tr>
<td>P4: R(x)b</td>
<td>P4: R(x)a</td>
</tr>
<tr>
<td>R(x)a</td>
<td>R(x)b</td>
</tr>
</tbody>
</table>

(a) (b)
Sequential Consistency

• weaker than strict/strong consistency
  • All operations are executed in some sequential order
  • each process issues operations in program order
    • Any valid interleaving is allowed
    • All agree on the same interleaving
    • Each process preserves its program order

Why is this weaker than strict/strong?
Sequential Consistency

- weaker than strict/strong consistency
  - All operations are executed in *some* sequential order
  - each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each process preserves its program order

• Why is this weaker than strict/strong?
• *Nothing is said about “most recent write”*
Linearizability
Linearizability

• Assumes sequential consistency and
  • If \( TS(x) < TS(y) \) then \( OP(x) \) should precede \( OP(y) \) in the sequence
  • Stronger than sequential consistency
  • Difference between linearizability and serializability?
    • Granularity: reads/writes versus transactions
Linearizability

• Assumes sequential consistency and
  • If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  • Stronger than sequential consistency
  • Difference between linearizability and serializability?
    • Granularity: reads/writes versus transactions

• Example:
  • Stay tuned...relevant for lock free data structures
  • Importantly: a property of concurrent objects
Causal consistency
Causal consistency

• Causally related writes seen by all processes in same order.
Causal consistency

- Causally related writes seen by all processes in same order.
  - Causally?
Causal consistency

- Causally related writes seen by all processes in same order.
- \textit{Causally}?

\textbf{Causal:}

If a write produces a value that causes another write, they are causally related.

\begin{verbatim}
X = 1
if(X > 0) {
    Y = 1
}
\end{verbatim}

Causal consistency $\rightarrow$ all see $X=1$, $Y=1$ in same order
Causal consistency

• Causally related writes seen by all processes in same order.
  • Causally?
Causal consistency

• Causally related writes seen by all processes in same order.
  • Causally?
  • Concurrent writes may be seen in different orders on different machines
Causal consistency

- Causally related writes seen by all processes in same order.
  - *Causally*
  - *Concurrent* writes may be seen in different orders on different machines

```
P1: W(x)a
P2: R(x)a  W(x)b
P3:         R(x)b  R(x)a
P4:         R(x)a  R(x)b
```
Causal consistency

• Causally related writes seen by all processes in same order.
  • *Causally*
  • *Concurrent* writes may be seen in different orders on different machines

P1: W(x)a
P2: R(x)a  W(x)b
P3:          R(x)b  R(x)a
P4:          R(x)a  R(x)b

(a)

Not permitted
Causal consistency

• Causally related writes seen by all processes in same order.
  • Causally?
  • Concurrent writes may be seen in different orders on different machines

Not permitted
Causal consistency

• Causally related writes seen by all processes in same order.
  • Causally?
  • Concurrent writes may be seen in different orders on different machines

P1: W(x)a
P2: R(x)a W(x)b
P3: R(x)b R(x)a
P4: R(x)a R(x)b

(a)

P1: W(x)a
P2: W(x)b
P3: R(x)b R(x)a
P4: R(x)a R(x)b

(b)

Not permitted  Permitted
Consistency models summary
## Consistency models summary

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strict</td>
<td>Absolute time ordering of all shared accesses matters.</td>
</tr>
<tr>
<td>Linearizability</td>
<td>All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp</td>
</tr>
<tr>
<td>Sequential</td>
<td>All processes see all shared accesses in the same order. Accesses are not ordered in time</td>
</tr>
<tr>
<td>Causal</td>
<td>All processes see causally-related shared accesses in the same order.</td>
</tr>
<tr>
<td>FIFO</td>
<td>All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>Consistency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weak</td>
<td>Shared data can be counted on to be consistent only after a synchronization is done</td>
</tr>
<tr>
<td>Release</td>
<td>Shared data are made consistent when a critical region is exited</td>
</tr>
<tr>
<td>Entry</td>
<td>Shared data pertaining to a critical region are made consistent when a critical region is entered.</td>
</tr>
</tbody>
</table>

(b)