An API For
Distributed Computing
(Building a TB-Scale Math Platform)

Cliff Click, CTO 0xdata
clifffc@0xdata.com
http://0xdata.com
http://clifffc.org/blog
H2O is...

- Pure Java, Open Source: 0xdata.com
  - https://github.com/0xdata/h2o/
- A Platform for doing Math
  - Parallel Distributed Math
  - In-memory analytics: GLM, GBM, RF, Logistic Reg
- Accessible via REST & JSON
- A K/V Store: ~150ns per get or put
- Distributed Fork/Join + Map/Reduce + K/V
A Collection of Distributed Vectors

// A Distributed Vector
// much more than 2billion elements
class Vec {
    long length(); // more than an int's worth

    // fast random access
    double at(long idx); // Get the idx'th elem
    boolean isNA(long idx);

    void set(long idx, double d); // writable
    void append(double d); // variable sized
}
Distributed Data Taxonomy

A Single Vector

Vec
A Very Large Single Vec

- Java primitive
- Usually `double`
- Length is a `long`
- `>>> 2^31` elements
- Compressed
  - Often 2x to 4x
  - Random access
- Linear access is FORTRAN speed
Distributed Data Taxonomy

A Single Distributed Vec

<table>
<thead>
<tr>
<th>Vec</th>
<th>JVM 1</th>
<th>JVM 2</th>
<th>JVM 3</th>
<th>JVM 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Heap</td>
<td>Heap</td>
<td>Heap</td>
<td>Heap</td>
</tr>
<tr>
<td></td>
<td>32Gig</td>
<td>32Gig</td>
<td>32Gig</td>
<td>32Gig</td>
</tr>
</tbody>
</table>

- Java Heap
  - Data In-Heap
  - Not off heap
  - Split Across Heaps

- GC management
  - Watch FullGC
  - Spill-to-disk
  - GC very cheap
  - Default GC

- Fortran-speed
- Java ease
Distributed Data Taxonomy

A Collection of Distributed Vecs

- Vecs aligned in heaps
- Optimized for concurrent access
- Random access any row, any JVM
- But faster if local... more on that later
Distributed Data Taxonomy

A Frame: Vec[]

- Similar to R frame
- Change Vecs freely
- Add, remove Vecs
- Describes a row of user data
- Struct-of-Arrays (vs ary-of-structs)
Distributed Data Taxonomy

A Chunk, Unit of Parallel Access

- Typically 1e3 to 1e6 elements
- Stored compressed
- In byte arrays
- Get/put is a few clock cycles including compression
Distributed Data Taxonomy

A Chunk[]: Concurrent Vec Access

- Access Row in a single thread
- Like a Java object
- Can read & write
- Both are full Java speed
- Conflicting writes: use JMM rules

```java
class Person {
    // fields
}
```
Distributed Data Taxonomy

Single Threaded Execution

- One CPU works a Chunk of rows
- Fork/Join work unit
- Big enough to cover control overheads
- Small enough to get fine-grained parallelism
- Map/Reduce
- Code written in a simple single-threaded style
Distributed Data Taxonomy

Distributed Parallel Execution

- All CPUs grab Chunks in parallel
- F/J load balances
- Code moves to Data
- Map/Reduce & F/J handles all sync
- H2O handles all comm, data manage
Distributed Data Taxonomy

Frame - a collection of Vecs
   Vec - a collection of Chunks
     Chunk - a collection of $10^3$ to $10^6$ elems
     elem - a java double

Row i - i'th elements of all the Vecs in a Frame
Distributed Coding Taxonomy

- No Distribution Coding:
  - Whole Algorithms, Whole Vector-Math
  - REST + JSON: e.g. load data, GLM, get results

- Simple Data-Parallel Coding:
  - Per-Row (or neighbor row) Math
  - Map/Reduce-style: e.g. Any dense linear algebra

- Complex Data-Parallel Coding
  - K/V Store, Graph Algo's, e.g. PageRank
Distributed Coding Taxonomy

- **No Distribution Coding:**
  - Whole Algorithms, Whole Vector-Math
  - REST + JSON: e.g. load data, GLM, get results

- **Simple Data-Parallel Coding:**
  - Per-Row (or neighbor row) Math
  - Map/Reduce-style: e.g. Any dense linear algebra

- **Complex Data-Parallel Coding**
  - K/V Store, Graph Algo's, e.g. PageRank

Read the docs!
This talk!
Join our GIT!
Simple Data-Parallel Coding

- Map/Reduce Per-Row: **Stateless**
  - Example from Linear Regression, $\Sigma y^2$

```java
double sumY2 = new MRTask() {
    double map( double d ) { return d*d; }
    double reduce( double d1, double d2 ) {
        return d1+d2;
    }
}.doAll( vecY );
```

- Auto-parallel, auto-distributed
- Fortran speed, Java Ease
Simple Data-Parallel Coding

- Map/Reduce Per-Row: **Statefull**
  - Linear Regression Pass 1: $\Sigma x, \Sigma y, \Sigma y^2$

```java
class LRPass1 extends MRTask {
    double sumX, sumY, sumY2; // I Can Haz State?
    void map( double X, double Y ) {
        sumX += X;  sumY += Y;  sumY2 += Y*Y;
    }
    void reduce( LRPass1 that ) {
        sumX  += that.sumX ;
        sumY  += that.sumY ;
        sumY2 += that.sumY2;
    }
}
```
Simple Data-Parallel Coding

- Map/Reduce Per-Row: **Batch Statefull**

```java
class LRPass1 extends MRTask {
    double sumX, sumY, sumY2;
    void map( Chunk CX, Chunk CY ) { // Whole Chunks
        for( int i=0; i<CX.len; i++ ) { // Batch!
            double X = CX.at(i), Y = CY.at(i);
            sumX += X;  sumY += Y;  sumY2 += Y*Y;
        }
    }
    void reduce( LRPass1 that ) {
        sumX  += that.sumX ;
        sumY  += that.sumY ;
        sumY2 += that.sumY2;
    }
}
```
Other Simple Examples

- Filter & Count (underage males):
  - (can pass in any number of Vecs or a Frame)

```java
long sumY2 = new MRTask() {
    long map( long age, long sex ) {
        return (age<=17 && sex==MALE) ? 1 : 0;
    }
    long reduce( long d1, long d2 ) {
        return d1+d2;
    }
}.doAll( vecAge, vecSex );
```
Other Simple Examples

- Filter into new set (underage males):
  - Can write or append subset of rows
    - (append order is preserved)

```java
class Filter extends MRTask {
    void map(Chunk CRisk, Chunk CAge, Chunk CSex) {
        for (int i=0; i<CAge.len; i++)
            if (CAge.at(i) <= 17 && CSex.at(i) == MALE)
                CRisk.append(CAge.at(i)); // build a set
    }
}
Vec risk = new AppendableVec();
new Filter().doAll(risk, vecAge, vecSex);
...risk... // all the underage males
```
Other Simple Examples

- Filter into new set (underage males):
  - Can write or append subset of rows
    - (append order is preserved)

```java
class Filter extends MRTask {
    void map(Chunk CRisk, Chunk CAge, Chunk CSex) {
        for (int i = 0; i < CAge.len; i++)
            if (CAge.at(i) <= 17 && CSex.at(i) == MALE)
                CRisk.append(CAge.at(i)); // build a set
    }
};
Vec risk = new AppendableVec();
new Filter().doAll( risk, vecAge, vecSex );
...risk... // all the underage males
```
Other Simple Examples

- Group-by: count of car-types by age

```java
class AgeHisto extends MRTask {
    long carAges[][]; // count of cars by age
    void map( Chunk CAge, Chunk CCar ) {
        carAges = new long[numAges][numCars];
        for( int i=0; i<CAge.len; i++ )
            carAges[CAge.at(i)][CCar.at(i)]++;
    }
    void reduce( AgeHisto that ) {
        for( int i=0; i<carAges.length; i++ )
            for( int j=0; i<carAges[i].length; j++ )
                carAges[i][j] += that.carAges[i][j];
    }
}
```
Other Simple Examples

- Group-by: count of car-types by age.

```java
class AgeHisto {
  long carAges[][],[]; // count of cars by age

  void map( Chunk CAge, Chunk CCar ) {
    carAges = new long[numAges][numCars];
    for ( int i=0; i<CAge.len; i++ )
      carAges[CAge.at(i)][CCar.at(i)]++;
  }

  void reduce( AgeHisto that ) {
    for ( int i=0; i<carAges.length; i++ )
      for ( int j=0; i<carAges[i].length; j++ )
        carAges[i][j] += that.carAges[i][j];
  }
}
```

Setting carAges in `map` makes it an **output** field. Private per-map call, single-threaded write access. Must be rolled-up in the **reduce** call.
Other Simple Examples

- **Uniques**
  - Uses distributed hash set

```java
class Uniques extends MRTask {
    DNonBlockingHashSet<Long> dnbhs = new ...;
    void map( long id ) { dnbhs.add(id); }
    void reduce( Uniques that ) {
        dnbhs.putAll(that.dnbhs);
    }
};
long uniques = new Uniques().
    doAll( vecVistors ).dnbhs.size();
```
Other Simple Examples

- Uniques
  - Uses distributed hash set

```java
class Uniques extends MRTask {
    DNonBlockingHashSet<Long> dnbhs = new DNonBlockingHashSet<Long>();
    void map(long id) { dnbhs.add(id); }
    void reduce(Uniques that) {
        dnbhs.putAll(that.dnbhs);
    }
}
long uniques = new Uniques().doAll(vecVistors).dnbhs.size();
```

Setting dnbhs in `<init>` makes it an input field. Shared across all maps(). Often read-only. This one is written, so needs a `reduce`.

Summary: Write (parallel) Java

- Most simple Java “just works”
- **Fast**: parallel distributed reads, writes, appends
  - Reads same speed as plain Java array loads
  - Writes, appends: slightly slower (compression)
  - Typically memory bandwidth limited
    - (may be CPU limited in a few cases)
- **Slower**: conflicting writes (but follows strict JMM)
  - Also supports transactional updates
Summary: Writing Analytics

• We're writing Big Data Analytics
  • Generalized Linear Modeling (ADMM, GLMNET)
    – Logistic Regression, Poisson, Gamma
  • Random Forest, GBM, KMeans++, KNN
• State-of-the-art Algorithms, running Distributed
• Solidly working on 100G datasets
  • Heading for Tera Scale
• Paying customers (in production!)
• Come write your own (distributed) algorithm!!!
Cool Systems Stuff...

- ... that I ran out of space for
- Reliable UDP, integrated w/RPC
- TCP is reliably UNReliable
  - Already have a reliable UDP framework, so no prob
- Fork/Join Goodies:
  - Priority Queues
  - Distributed F/J
  - Surviving fork bombs & lost threads
- K/V does JMM via hardware-like MESI protocol
H2O is...

- Pure Java, Open Source: 0xdata.com
  - https://github.com/0xdata/h2o/
- A Platform for doing Math
  - Parallel Distributed Math
  - In-memory analytics: GLM, GBM, RF, Logistic Reg
- Accessible via REST & JSON
- A K/V Store: ~150ns per get or put
- Distributed Fork/Join + Map/Reduce + K/V
The Platform

JVM 1
- extends MRTask
- extends DRemoteTask
- extends DTask
- extends Iced
- AutoBuffer
- byte[]
- NFS
- HDFS

K/V get/put

UDP / TCP

JVM 2
- extends MRTask
- extends DRemoteTask
- extends DTask
- extends Iced
- AutoBuffer
- byte[]
- NFS
- HDFS
TCP Fails

• In <5mins, I can force a TCP fail on Linux
• "Fail": means Server opens+writes+closes
  • NO ERRORS
  • Client gets no data, no errors
  • In my lab (no virtualization) or EC2
• Basically, H2O can mimic a DDOS attack
  • And Linux will "cheat" on the TCP protocol
  • And cancel valid, in-progress, TCP handshakes
  • Verified w/wireshark
TCP Fails

- Any formal verification? (yes lots)
- Of recent Linux kernals?
  - Ones with DDOS-defense built-in?