CS 6240: Parallel Data Processing in MapReduce

Mirek Riedewald

Custom Key and Value Types

- Option 1: encode them as text strings
  - Use existing Text class
  - Encoding/decoding a bit clumsy and expensive
- Option 2: define new Writable or WritableComparable
- Option 3: use ArrayWritable
  - Elements must all be instances of the same class

Creating New Hadoop Value Types

- Implement **Writable** interface
  - Implement readFields(DataInput) method
    - Deserializes fields of this object from input
  - Implement write(DataOutput) method
    - Serializes fields of this object to output
- Typical implementation also has static read(DataInput) method that calls readFields()

```
public class MyWritable implements Writable {
    // Some data
    private int counter;
    private long timestamp;

    // Default constructor to allow (de)serialization
    MyWritable() { }

    public void write(DataOutput out) throws IOException {
      out.writeInt(counter);
      out.writeLong(timestamp);
    }

    public void readFields(DataInput in) throws IOException {
      counter = in.readInt();
      timestamp = in.readLong();
    }

    public static MyWritable read(DataInput in) throws IOException {
      MyWritable w = new MyWritable();
      w.readFields(in);
      return w;
    }
  }
```

Example from Hadoop API

Creating New Hadoop Key Types

- Implement **WritableComparable** interface
  - Implement Writable interface
    - Methods readFields() and write()
  - Implement Comparable interface
    - Method compareTo()

Example from White's Hadoop book

```
import java.io.*; import org.apache.hadoop.io.*;

public class IntPair implements WritableComparable<IntPair> {
  private int first;  private int second;

  public IntPair() {}
  public IntPair(int first, int second) { set(first, second); }
  public void set(int first, int second) { this.first = first; this.second = second; }
  public int getFirst() { return first; }
  public int getSecond() { return second; }

  @Override
  public void write(DataOutput out) throws IOException {
    out.writeInt(first);  out.writeInt(second);  }

  @Override
  public void readFields(DataInput in) throws IOException {
    first = in.readInt();  second = in.readInt();  }

  @Override
  public int hashCode() { return first * 163 + second; }

  @Override
  public boolean equals(Object o) {
    if (o instanceof IntPair) {
      IntPair ip = (IntPair) o;
      return first == ip.first && second == ip.second;
    }
    return false;
  }

  @Override
  public String toString() { return first + "\t" + second; }

  @Override
  public int compareTo(IntPair ip) {
    int cmp = compare(first, ip.first);
    if (cmp != 0) {
      return cmp;
    }
    return compare(second, ip.second);
  }

  /**
   * Convenience method for comparing two ints.
   */
  public static int compare(int a, int b) {
    return (a < b ? -1 : (a == b ? 0 : 1));
  }
}
```