Announcements

- Cookies: we are behind. Sign up. 4% of your grade
  - now visible on grades.cs.umd.edu
- Project 6 grades posted on grades.cs.umd.edu
- Project 8: Map Reduce, part 1
  - both tracks
  - due April 29th
- Project 9: Map Reduce, part 2
  - separate tracks
  - due May 6th
Open source contribution

• Some people still haven't gotten started
  • find a project with recent updates, recent bug tracker activity, active source code repository
  • build from source
  • count lines of code

• You are responsible for getting this done by the last day of class, regardless of any feedback from us or lack of it
In class discussion and lab

If you have Eclipse open, you must close project 8
Right click on project 8, select close project

You may not open any project files during class
Part 1

- FileSplitCalculator
- Reader
- AccumulatingPairWriter
- InMemoryMapReduce
- WorkerRunner
- HeartBeat
Given a requested number of splits and an array of files, produce a list of FileRanges that split the files into reasonably sized ranges. No range should be larger than 1+totalSize/minSplits bytes, and the total number of splits returned should be no longer than minSplits+#files-1

```java
public static List<FileRange> splitFiles(
    int minSplits, File... files) { ... }
```
FileSplitCalculator

- Example: 3 files
  - file1: 20,000 bytes
  - file2: 7,000 bytes
  - file3: 13,000 bytes
- Asked for 5 min splits
- What is maximum size of a split?
- What is maximum number of splits?
- What are example splits
Given a FileRange, invoke the write method of an PairWriter for each line of text starting in the FileRange, with a FileIndex corresponding to the index of the first character of the line.

```java
public static void read(FileRange range,
PairWriter<FileIndex, String> writer) {
    ... }
```
Reader

- Use RandomAccessFile
  - use seek(long) to position reader
  - use getFilePointer() to get current position
  - use readLine() to read from current position to EOL
  - do not use readChar() - reads 2 bytes as 16 bit character
- Several ways to handle skipping partial lines
AccumulatingPairWriter

- `PairWriter<K,V>` than constructs a `Map<K,List<V>>`
AccumulatingPairWriter

• Hopefully straightforward
InMemoryMapReduce

• Perform Map reduce in memory
• Create AccumulatingPairWriter
• For each File, construct one FileRange
• For each FileRange, invoke Reader that passes results to mapper that passes results to AccumulatingPairWriter
• For each Key, List<Value> in AccumulatingPairWriter
  • invoke reducer
• writing results to file in output directory
InMemoryMapReduce

Files → FileRanges → Reader

Accumulating PairWriter

Mapper Adapter

Reducer

ObjectOutputStream PairWriter
WorkerRunner

• Similar to multicast exercise
• Create Server socket
• Loop:
  • broadcast availability
  • wait up to 1 second for a request
    • if you get one, handle it
Notes

- Change Constants.MULTICAST_PORT
- Use Multicast.getDefault()
- create one server socket and reuse it for each call to accept (thereby reusing the same port)
Handling a Worker request

- Accept connection
- read Worker via ObjectInputStream
- set up HeartBeat
- run worker
  - if completes normally, call done on heart beat
  - if throws exception, call fail on heart beat
- close socket
HeartBeat

- Utility class for checking a remote heart beat.

- On one side, creating a HeartBeat will send a stream of 0's over a socket, one per second, terminating when either the done() or fail() methods are called.

- On the other wise, waitForSuccess(Socket) will return normally when the other side calls done(), and by throwing an exception if the other side calls fail(), the connection is closed, or no heart beat is received for for 10 seconds.
Open discussion