CMSC 433

Map Reduce exercise
April 20th, 2010
Announcements

• Cookies: we are behind. Sign up.
• Project 8: Map Reduce, part 1
  • both tracks
  • due April 29th
• Project 9: Map Reduce, part 2
  • separate tracks
  • due May 6th
Map reduce project

- Somewhat simplified/different
- No IntWritable/TextWritable, just serializable objects
- Contexts are simplified
Part 1

- FileSplitCalculator
- Reader
- AccumulatingPairWriter
- InMemoryMapReduce
- WorkerRunner
- HeartBeat
Part 2, common

- CombiningPairWriter
- Some additional Map Reduce algorithms, with combiners
Part 2, core mastery

- Multithreaded, non-distributed, map-reduce
Part 2, normal track

- DistributedMapReduce (~300 lines of code)
- MapWorker (~130 lines of code)
- ReduceWorker (~200 lines of code)
Part I
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 \(1+\text{totalSize}/\text{minSplits}\) bytes, and the total number of splits returned should be no longer than \(\text{minSplits}+#\text{files}-1\).

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

- 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) {
    ... }
```
AccumulatingPairWriter

- `PairWriter<K,V>` than constructs a `Map<K,List<V>>`
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
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
Handling a Worker request

• Accept connection
• read Worker via ObjectInputSteam
• 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.