Distributed Computing with Ruby

Background: What is MapReduce?

The Algorithm:

1. A large data set is divided into records.
2. Each record is passed through a *Mapper,* which may do anything to the data, and emit key/value pairs.
3. Mapper output is sorted by key
4. Each set of values for a key is passed to a Reducer, which may do anything with this collection.

Background: What is MapReduce?

The Algorithm:

1. A large data set is divided into records.
2. Each record is passed through a *Mapper,* which may do anything to the data, and emit key/value pairs.
3. Mapper output is sorted by key
4. The *Reducer* is called once per key, and is passed all the values for that key. It may do anything to this data.

An Example:

1. Server log data is parsed into events.
2. A Mapper filters for search events, geocodes them using the IP address, and puts out location/search term pairs.
3. Mapper output is sorted by location.
4. A Reducer is called once per location, and performs frequency analysis on search terms for that location.

Background: What is Hadoop?

Hadoop is...

- ...an implementation of MapReduce in Java, with a Java callback API.
- ...an Apache project.
- ...bundled with a distributed, redundant filesystem that allows for reliably storing and scalably processing petabytes of data.
- ...used by Facebook (server log analysis), NYT (batch image processing), Yahoo! (100K CPUs supporting search and ads), and other companies you've heard of.
- ...meant to be used with commodity hardware, and scaled by adding cheap servers.

Background: Hadoop Streaming

Hadoop Streaming is an API wrapper lets you run MapReduce jobs without using the Java callback API. Any language that can handle STDIN and STDOUT can be used with Hadoop!

Why is this good?

- Java dev tools are a PITA
- Ruby rocks
- Code re-use
- Prototyping and rapid development

Example 1: Word Count

Mapper:

```
#!/usr/bin/env ruby
STDIN.each_line do |line|
  word_count = {}
  line.split.each do |word|
    word_count[word] ||= 0
    word_count[word] += 1
  end

  word_count.each do |k,v|
    puts "#{k}\t#{v}"
  end
end
```

Reducer:

```
#!/usr/bin/env ruby

curr_word = nil
tally = 0
STDIN.each_line do |line|
  word, count = line.strip.split
  if word != curr_word
    puts "#{curr_word}\t#{tally}"
unless curr_word.nil?
    curr_word = word
    tally = 0
  end

  tally += count.to_i
end

puts "#{curr_word}\t#{tally}" unless
curr_word.nil?
```

Example 1: Word Count

Input:

```
Stately, plump Buck Mulligan came
from the stairhead, bearing a bowl
of
lather on which a mirror and a razor
lay crossed. A yellow dressinggown,
ungirdled, was sustained gently
behind him on the mild morning air.
He
held the bowl aloft and intoned:

--_Introibo ad altare Dei_.

Halted, he peered down the dark
winding stairs and called out
coarsely:
```

…

Output:

```
...
aside     18
ask       39
askance   4
asked     86
askew     3
askew,    1
asking    23
asks      5
aslant    2
asleep    11
aspect    1
aspergills      1
asperity          2
asphalted         1
asphyxiating      1

...
```

Ex. 2: Genetic Traveling Salesman

Implementing a non-toy problem with Ruby and Hadoop: A Genetic Algorithm solver of the Traveling Salesman Problem

- GAs are very parallelizable.
- Traveling Salesman is well-studied, and susceptible to GA methods

Ex.2: Traveling Salesman in a Nutshell

You have N cities, randomly placed about a map. What is the shortest route that covers them all?

Harder than it sounds:

- 5 cities ➡ 120 possible solutions
- 10 cities ➡ 3,628,800 possible solutions
- 15 cities ➡ 1,307,674,368,000 possible solutions
- 20 cities ➡ 2.433*10^18 possible solutions

Best known approach to exact solutions gives 2^n performance - BUT there are heuristic and inexact methods, including...

Ex. 2: Genetic Algorithms in a Nutshell

Genetic Algorithms evolve a solution by mimicking the process of natural selection.

- Possible values in a solution space are modeled as *genes.*
- A collection of genes representing one solution is modeled as an *organism.*
- A large population of organisms is generated randomly, then judged with a *fitness function.*
- A selection of the fittest individuals (plus a few less fit ones) are bred and mutated.
- Lather, rinse, repeat.

By keeping the fittest individuals and introducing randomness, the population drifts toward maxima in the solution space.

Ex. 2: Traveling Salesman Organisms

My Hadoop filesystem is host to a huge number of these:

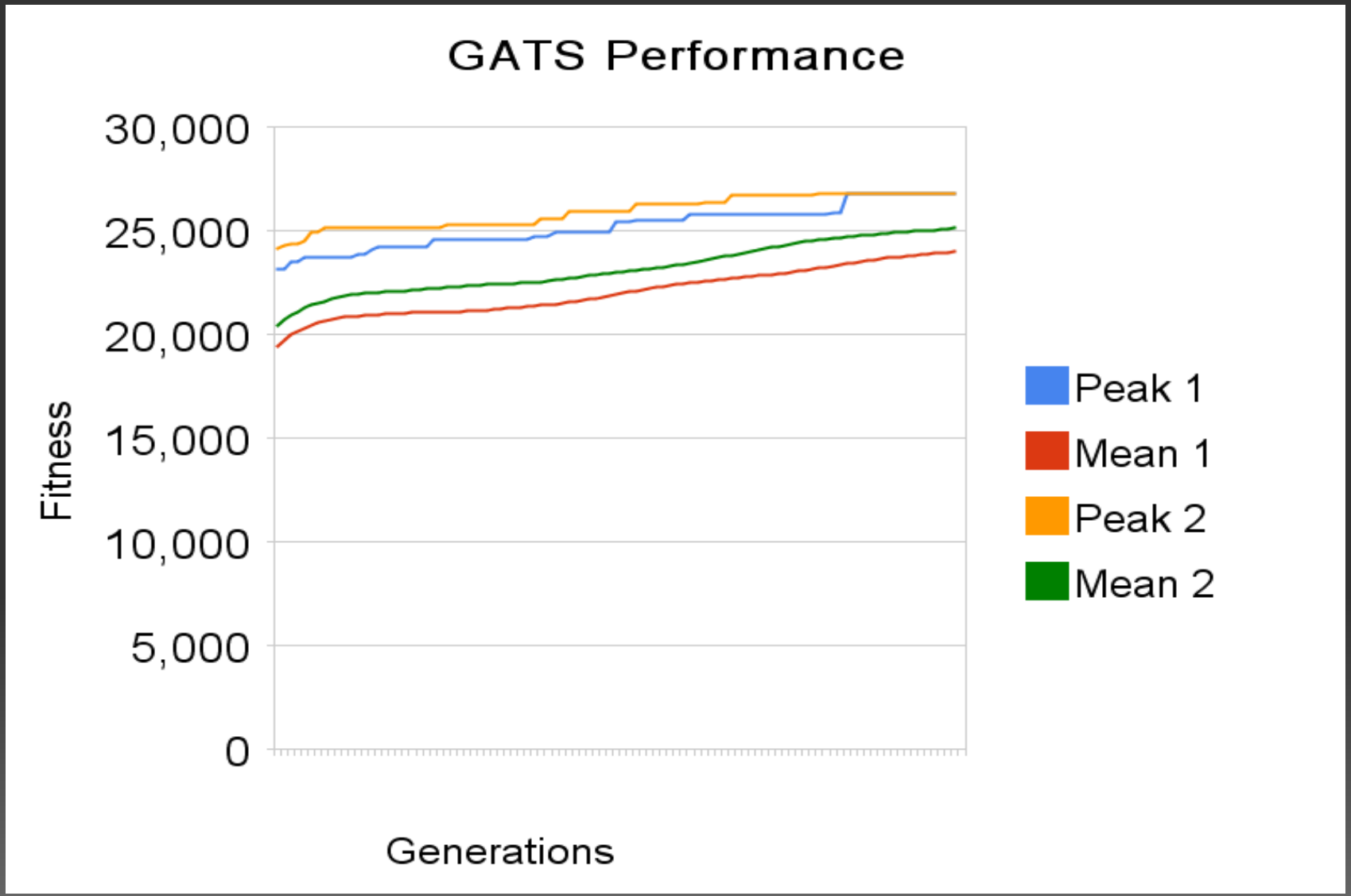
```
...
14:9:7:12:15:14:12:1:0:2:2:8:2:5:2:1:1:1:0:0    21865.5163525769
14:9:7:3:11:9:2:9:0:1:3:3:3:6:4:1:1:0:1:0       21918.7523231313
15:0:0:7:15:12:12:6:5:1:8:5:6:3:4:2:2:1:1:0     19459.585680597
15:0:15:12:5:1:11:0:5:1:5:4:1:3:0:4:3:0:0:0     20583.7927333763
15:0:15:15:4:7:7:3:8:4:9:5:1:0:0:3:1:0:0:0      21247.3207182743
15:0:3:6:3:13:2:1:2:10:3:6:2:3:1:4:1:1:1:0      20637.9513028541
...
```

Each record (organism) has a collection of genes and a fitness score.

Ex. 2: Fitness Mapper for GATS

```
def process(line)
  fields = line.split "\t"
  if ('x' == fields[1]) # no fitness yet
    genome = fields[0]
    city_indices = get_cities_from_genes genome.split(':')
    fitness = 0
    # "Fitness" is a minimum distance (here mapped to a maximum score)
    (city_indices.size - 1).times do |i|
      fitness += dist(TravSales::CITIES[city_indices[i].to_i], TravSales::
CITIES[city_indices[i + 1].to_i])
    end
    fitness = 1500 * TravSales::CITIES.size - fitness
    yield [genome + "\t" + fitness.to_s]
  else
    yield [line]
  end
```

Ex. 2: GATS Results



JRuby Experiments

Why JRuby?

- Performance - factor of 3 better than Ruby 1.8.6
- Java Integration - Is it possible to hook directly into the API, and bypass Hadoop Streaming overhead?

Results: Performance is improved as expected (sometimes after a bit of code tweaking). Java Integration, on the other hand...

Limits of JRuby Integration

```
class WordCountJRuby
  class TestMapper < MapReduceBase
    include org.apache.hadoop.mapred.Mapper

    def map(key, value, output, reporter)
      line = value.toString
      tokenizer = StringTokenizer.new line
      while tokenizer.hasMoreTokens
        output.collect Text.new(tokenizer.nextToken), IntWritable.new(1)
      end
    end
  end

  def self.main(args)
    conf = JobConf.new
    conf.setMapperClass(TestMapper) # FAIL
  end
end
```

Can't inherit a Java class, then pass it back to JVM. :-(

Conclusions

- Probably not ready for production work on extremely large data sets (though JRuby helps with performance issues).
- When JRuby integration issues are solved, that may change.
- Fantastic option for prototyping & rapid development
- On small (<5 nodes) clusters, best suited to data sets <10^9 records

Links

- This talk: http://bradheintz.com/talks
- Blog: http://kickasslabs.com/
- Hadoop: http://hadoop.apache.org/
- MapReduce: http://labs.google.com/papers/mapreduce.html
- Wukong: http://github.com/mrflip/wukong
- JRuby: http://jruby.codehaus.org/
- Read more about Genetic Algorithms and Traveling Salesman Problem on Wikipedia

Thanks

- Philip "Flip" Kromer, for Wukong
- Headius, Lopex and the JRuby crew for help with my bugs
- NYC.rb for hosting
- You