EMR 101

Two possible audiences “Programmer/Dev” vs. “IT/Ops”, will try to address both.

“Pre” Basics

- EMR = Elastic Map Reduce
- Amazon’s “Hadoop as a service”
- Standard Amazon “pay for what you use” model

Costs: EC2 + Transfer + Storage.
Amazon charges approx 25% of on demand cost as overhead on EC2 part of the costs.

Hadoop Basics

- Used to solve "embarrassingly parallel" problems
- Move computation to data
- Requires only two fairly simple methods ("map" and "reduce"), or higher level language (Pig, Hive)

This is the "to drive a car you have to take it apart and reassemble it first" theory of learning something.
Embarrassingly parallel: able to work on each part 100% independent of all other parts.
Move computation to data: what I see as the "lightbulb" of Hadoop. Speed is CPU>RAM>Disk/Network, so moving a program through the network to the data is key. Hadoop is fancier than RPC since what computation is done is dynamic at runtime
Pig = Scripting, Hive = SQL. I prefer Pig due to the ability to "globally optimize".

Hadoop: Data Flow

- Split: takes a description of the input, and divides it into small chunks. input=>(k,v)
- Map: takes a split and applies the map function. (k,v) => (k', v')
- Shuffle/Sort: groups all k's. (k', (v1,v2,v3...,vN))
- Reduce: applies reduce function to (k', list)

Split: "v" is meta data (e.g. a pointer to the data).
Shuffle is the poorly named most important step, and you get it for free.
Shuffle produces the powerful guarantee that k' is globally unique for the reducer to operate on.

Hadoop: System

- JobTracker: Manages jobs, creates splits. One node.
- Name/SecondaryName: HDFS meta data. One node.
- TaskTracker/Data: "Workers". N nodes.

Not terrible to self manage, 5 process with a shared config. Only 2 types of boxes (usually), and they are 1 vs N.
Split: runs on the JobTracker. BonusQ: why does split produce meta data? Bottleneck otherwise (easy to say take 100MB at offset 1GB, hard to push 100MB around)
HDFS: Important, but behind the scenes. Redundancy gives safety, but also choices for Job to "move computation to data".

Hadoop: Examples

Example 1: Play act hadoop (good for ops, bad for dev)
Example 2: Letter count (bad for ops, good for dev)

Hadoop <=> EMR

- Job/Name/SecondaryName <=> Master
- Task/Data <=> Core
- Task Only <=> Task

Why both core & task? Cost optimization (if your use case works, you can have a 24x7 core, and scale up tasks)

S3

- EMR is an island
- S3 is the bridge in and out (Dynamo...)
- Plan ahead, IO is the hardest to scale in Amazon

IO: I mean in/out of S3 itself.
Within EMR, S3 scales IO great.

Starting EMR

Show GUI, show CLI.

Monitoring EMR

- GUI
  - Debug Button. S3 logs.
- CLI
  - ssh hadoop@master
  - /mnt/var/log
  - lynx http://localhost:9100/

Advice/Tips

- Spot instances
- S3/EC2/EBS IO issues
- Tuning... beyond 101

Spot Instances: Blessing and curse. Can save enormously by taking termination risk, and Hadoop is designed to recover from failure. Problem? When you start to lose instances, you lose a _lot_ of them (usually, all). Plus, there are periodic price jumps, and when you rely on artificially cheap rates, the "real" rate seems astronomically high.
Getting data into S3, if you have an always updating data source, can be challenging with Amazon's IO. Medium/Large seems to cap at 100Mb. XL is at least 1Gb (we saw over 3Gb, so maybe 10Gb?). Once in S3, you're golden.
Tuning: Beyond 101. But, beware pig's default of one reducer!
Lessons learned: don't install packages from original source, put in S3 (people will hate you if you spin up a 300 node cluster and install an RPM on all nodes!)
ask for more capacity ahead of time (you have a limit on number of EC2 boxes you can start).