HadoopIsNot

What Hadoop is Not

We see a lot of emails where people hear about Hadoop, and think it will be the silver bullet to solve all their application/datacentre problems. It is not. It solves some specific problems for some companies and organisations, but only after they have understood the technology and where it is appropriate. If you start using Hadoop in the belief it is a drop-in replacement for your database or SAN filesystem, you will be disappointed.

Apache Hadoop is not a substitute for a database

Databases are wonderful. Issue an SQL SELECT call against an indexed/tuned database and the response comes back in milliseconds. Want to change that data? SQL UPDATE and the change is in. Hadoop does not do this.

Hadoop stores data in files, and does not index them. If you want to find something, you have to run a MapReduce job going through all the data. This takes time, and means that you cannot directly use Hadoop as a substitute for a database. Where Hadoop works is where the data is too big for a database (i.e. you have reached the technical limits, not just that you don't want to pay for a database license). With very large datasets, the cost of regenerating indexes is so high you can't easily index changing data. With many machines trying to write to the database, you can't get locks on it. Here the idea of vaguely-related files in a distributed filesystem can work.

There is a high performance column-table database that runs on top of Hadoop HDFS: Apache HBase. This is a great place to keep the results extracted from your original data.

MapReduce is not always the best algorithm

MapReduce is a profound idea: taking a simple functional programming operation and applying it, in parallel, to gigabytes or terabytes of data. But there is a price. For that parallelism, you need to have each MR operation independent from all the others. If you need to know everything that has gone before, you have a problem. Such problems can be aided by

- Iteration: run multiple MR jobs, with the output of one being the input to the next.
- Shared state information. HBase is an option to consider here, otherwise something like memcache is an option.

Do not try to remember things in shared variables, as they are only remembered in a single JVM, for the life of that JVM. That is the wrong way to work in a massively parallel environment.

Hadoop and MapReduce is not a place to learn Java programming

There are currently a lot of assumptions in the Hadoop APIs and documentation, assumptions that you know the basics of Java programming, and of the common error messages you get when things don't work. If you do not know about classpaths, how to compile and debug Java code, step back from Hadoop and learn a bit more about Java before proceeding.

Hadoop is not an ideal place to learn networking error messages

You will find things work a lot easier if you are already familiar with networking and the common error messages -for example, what "Connection Refused" means, and how is different from "No Route to Host".

A lot of people post onto the user list with problems related to "Connection Refused", "No Route to Host" and other common TCP-IP level errors. These are usually signs of an invalid cluster configuration, some parts of the cluster not running, or machines not being able to talk to each other on the LAN. People on the mailing list cannot debug your network configuration for you, as it is your network, not theirs. They can point you at some of the tools and tests to try, but since it will take a day for every email round trip, you won't find this a very fast way to get help.

Nobody in the Hadoop team are deliberately trying to make things hard, its just that when things do not work in a large distributed system, you get some interesting error messages. If you can help improve those network messages or diagnostics, we would love to have that code.

Hadoop clusters are not a place to learn Unix/Linux system administration

You need to know your way round a Unix/Linux system. How to install it, what the various files in /etc/ are for, how to set up networking, what is a good hosts table, how to debug DNS problems, why to keep logs on a separate disk from the root disk, etc. If you cannot look after a single machine, you aren't going to be able to handle a cluster of 80 of them. That said, don't try maintaining those 80+ boxes using the same technique of hand-editing files like HadoopIsNot etc/hosts, because it doesn't scale.

Things you need to know

- SSH, what it is, how to set up authorized_keys, how to use ssh and scp
- ifconfig, nslookup and other network config/diagnostics tools
- How your platform keeps itself up to date
- What the various log files your machine generates, and what they mean
- How to set up native filesystems and mount them

This is important. If you don't know these, you are out of your depth and should not start installing Hadoop until you have the basics of a couple of Linux systems up and running, letting you ssh in to each of them without entering a password, know each other's hostname and such like. The Hadoop installation documents all assume you can do these things, and aren't going to bother explaining about them.

HDFS is not a complete POSIX filesystem

The Posix filesystem model has expectations of what a filesystem can do that HDFS cannot. A key one is that you can only add data to the end of the file, not seek to the middle and write things. You cannot seamlessly map code that assumes that all filesystems are Posix-compatible to HDFS.