OldHadoopTutorial

How to Setup Nutch (V1.1) and Hadoop

Note: Originally this (NutchHadoopTutorial0.8) was written for version 0.8 of Nutch. This has been edited by people other than the original author so statements like "I did this" or "I recommend that" are slightly misleading.

By default, out of the box, Nutch runs in a single process on one machine. This may suit you fine if you have a small site to crawl and index, but most people choose Nutch because of its capability to run on a Hadoop cluster. This gives you the benefit of a distributed file system (HDFS) and MapReduce processing style. The purpose of this tutorial is to provide a step-by-step method to get Nutch running with Hadoop file system on multiple machines, including being able to both index (crawl) and search across multiple machines.

This document does not go into the Nutch or Hadoop architecture. It only tells how to get the systems up and running. At the end of the tutorial though I will point you to relevant resources if you want to know more about the architecture of Nutch and Hadoop.

The tutorial comes in two phases. Firstly we get Hadoop running on a single machine (a bit of a simple cluster!) and then more than one machine.

Assumptions

Some things are assumed for this tutorial:

First: I performed some setup and using root level access. This included setting up the same user across multiple machines and setting up a local filesystem outside of the user's home directory. Root access is not required to setup Nutch and Hadoop (although sometimes it is convenient). If you do not have root access, you will need the same user setup across all machines which you are using and you will probably need to use a local filesystem inside of your home directory.

Two: all boxes will need an SSH server running (not just a client) as Hadoop uses SSH to start slave servers. Although we try to explain how to set up ssh so that communication between machines does not require a password you may need to learn how to do that elsewhere.

Three: This tutorial uses Whitebox Enterprise Linux 3 Respin 2 (WHEL). For those of you who don't know Whitebox, it is a RedHat Enterprise Linux clone. You should be able to follow along for any linux system, but the systems I use are Whitebox. (Later versions of this document have been tested using Ubuntu Linux, but as before

Four: This tutorial was originally written for Nutch 0.8 Dev Revision 385702, but has been updated to work with Nutch 1.1RC. It may not be compatible with future releases of either Nutch or Hadoop.

Five: For this tutorial we setup nutch across 6 different computers. If you are using a different number of machines you should still be fine but you should have at least two different machines to prove the distributed capabilities of both HDFS and MapReduce.

Six: Remember that this is a tutorial from my personal experience setting up Nutch and Hadoop. If something doesn't work for you try searching and sending a message to the Nutch or Hadoop users mailing list. Suggestions or tips are welcome. Why not add them to the end of this Wiki page?

Seven: We assume that you are a Java programmer familiar with the concepts of JAVA_HOME, ant build tool, subversion, IDEs and such like.

Our Network Setup

First let me layout the computers that we used in our setup. To setup Nutch and Hadoop we had 7 commodity computers ranging from 750Mghz to 1.0 Ghz. Each computer had at least 128 Megs of RAM and at least a 10 Gigabyte hard drive. One computer had dual 750 Mghz CPUs and another had dual 30 Gigabyte hard drives. All of these computers were purchased for under $500.00 at a liquidation sale. I am telling you this to let you know that you don't have to have big hardware to get up and running with Nutch and Hadoop. Our computers were named like this:

```
devcluster01
devcluster02
devcluster03
devcluster04
devcluster05
devcluster06
```

Our master node was devcluster01. By master node I mean that it ran the Hadoop services that coordinated with the slave nodes (all of the other computers) and it was the machine on which we performed our crawl and deployed our search website.

Downloading Nutch and Hadoop

Both Nutch and Hadoop are downloadable from the Apache website. The necessary Hadoop files are bundled with Nutch so unless you are going to be developing Hadoop you only need to download Nutch.

We built Nutch from source after downloading it from its subversion repository. Nightly builds of Nutch can be found here:

http://hudson.zones.apache.org/hudson/job/Nutch-trunk/

At time of writing this version (Jun 2010) Nutch includes Hadoop Jars version 0.20.2

You can get a packaged tarball or extract from subversion. Knowing how to use tar or subversion is outside of the scope of this tutorial. Once you have a subversion client you can either browse the Nutch subversion webpage at:

http://nutch.apache.org/version_control.html

Or you can access the Nutch subversion repository through the client at:

http://svn.apache.org/repos/asf/nutch/ (previously at http://svn.apache.org/repos/asf/lucene/nutch/ when Nutch was a part of Lucene)

We are going to use ant to build it so if you have java and ant installed you should be fine.

I am not going to go into how to install java or ant, if you are working with this level of software you should know how to do that and there are plenty of tutorial on building software with ant. If you want a complete reference for ant pick up Erik Hatcher's book "*Java Development with Ant*":

http://www.manning.com/hatcher

It is worth noting that previous versions of Nutch came already built. But nowadays the release is just source code and so does have to be built before use.

Building Nutch and Hadoop

Once you have Nutch downloaded and unpacked look inside it where you should see the following folders and files:

```
+ bin
+ conf
+ docs
+ lib
+ site
+ src
    build.properties (add this one)
    build.xml
    CHANGES.txt
    default.properties
    index.html
    LICENSE.txt
    README.txt
```

Add a build.properties file and inside of it add a variable called dist.dir with its value being the location where you want to build nutch. So if you are building on a linux machine it would look something like this:

```
dist.dir=/path/to/build
```

This step is actually optional as Nutch will create a build directory inside of the directory where you unzipped it by default, but I prefer building it to an external directory. You can name the build directory anything you want but I recommend using a new empty folder to build into. Remember to create the build folder if it doesn't already exist.

To build nutch call the package ant task like this:

```
ant package
```

This should build nutch into your build folder. When it is finished you are ready to move on to deploying and configuring nutch.

Setting Up The Deployment Architecture

Once we get nutch deployed to all six machines we are going to call a script called start-all.sh that starts the services on the master node and data nodes. This means that the script is going to start the hadoop daemons on the master node and then will ssh into all of the slave nodes and start daemons on the slave nodes.

The start-all.sh script is going to expect that nutch is installed in exactly the same location on every machine. It is also going to expect that Hadoop is storing the data at the exact same filepath on every machine.

The way we did it was to create the following directory structure on every machine. The search directory is where Nutch is installed. The filesystem is the root of the hadoop filesystem. The home directory is the nutch users's home directory. On our master node we also installed a tomcat 5.5 server for searching.

```
/nutch
  /search
    (nutch installation goes here)
  /filesystem
  /local (used for local directory for searching)
  /home
    (nutch user's home directory)
  /tomcat    (only on one server for searching)
```

I am not going to go into detail about how to install Tomcat as again there are plenty of tutorials on how to do that. I will say that we removed all of the wars from the webapps directory and created a folder called ROOT under webapps into which we unzipped the Nutch war file (nutch-0.8-dev.war). This makes it easy to edit configuration files inside of the Nutch war

So log into the master nodes and all of the slave nodes as root. Create the nutch user and the different filesystems with the following commands:

```
ssh -l root devcluster01

mkdir /nutch
mkdir /nutch/search
mkdir /nutch/filesystem
mkdir /nutch/local
mkdir /nutch/home

groupadd users
useradd -d /nutch/home -g users nutch
chown -R nutch:users /nutch
passwd nutch nutchuserpassword
```

Again if you don't have root level access you will still need the same user on each machine as the start-all.sh script expects it. It doesn't have to be a user named nutch user although that is what we use. Also you could put the filesystem under the common user's home directory. Basically, you don't have to be root, but it helps.

The start-all.sh script that starts the daemons on the master and slave nodes is going to need to be able to use a password-less login through ssh. For this we are going to have to setup ssh keys on each of the nodes. Since the master node is going to start daemons on itself we also need the ability to user a password-less login on itself.

You might have seen some old tutorials or information floating around the user lists that said you would need to edit the SSH daemon to allow the property PermitUserEnvironment and to setup local environment variables for the ssh logins through an environment file. This has changed. We no longer need to edit the ssh daemon and we can setup the environment variables inside of the hadoop-env.sh file. Open the hadoop-env.sh file inside of vi:

```
cd /nutch/search/conf
vi hadoop-env.sh
```

Below is a template for the environment variables that need to be changed in the hadoop-env.sh file:

```
export HADOOP_HOME=/nutch/search
export JAVA_HOME=/usr/java/jdk1.5.0_06
export HADOOP_LOG_DIR=${HADOOP_HOME}/logs
export HADOOP_SLAVES=${HADOOP_HOME}/conf/slaves
```

There are other variables in this file which will affect the behavior of Hadoop. If when you start running the script later you start getting ssh errors, try changing the HADOOP_SSH_OPTS variable. Note also that, after the initial copy, you can set HADOOP_MASTER in your conf/hadoop-env.sh and it will use rsync changes on the master to each slave node. There is a section below on how to do this.

Next we are going to create the keys on the master node and copy them over to each of the slave nodes. This must be done as the nutch user we created earlier. Don't just su in as the nutch user, start up a new shell and login as the nutch user. If you su in the password-less login we are about to setup will not work in testing but will work when a new session is started as the nutch user.

```
cd /nutch/home

ssh-keygen -t rsa (Use empty responses for each prompt)
  Enter passphrase (empty for no passphrase):
  Enter same passphrase again:
  Your identification has been saved in /nutch/home/.ssh/id_rsa.
  Your public key has been saved in /nutch/home/.ssh/id_rsa.pub.
  The key fingerprint is:
  a6:5c:c3:eb:18:94:0b:06:a1:a6:29:58:fa:80:0a:bc nutch@localhost
```

On the master node you will copy the public key you just created to a file called authorized_keys in the same directory:

```
cd /nutch/home/.ssh
cp id_rsa.pub authorized_keys
```

You only have to run the ssh-keygen on the master node. On each of the slave nodes after the filesystem is created you will just need to copy the keys over using scp. eg to send the authorisation from to devcluster02 we might do this on devcluster01

```
scp /nutch/home/.ssh/authorized_keys nutch@devcluster02:/nutch/home/.ssh/authorized_keys
```

You will have to enter the password for the nutch user the first time. An ssh prompt will appear the first time you login to each computer asking if you want to add the computer to the known hosts. Answer yes to the prompt. Once the key is copied you shouldn't have to enter a password when logging in as the nutch user. Test it by logging into the slave nodes that you just copied the keys to:

```
ssh devcluster02
nutch@devcluster02$ (a command prompt should appear without requiring a password)
hostname (should return the name of the slave node, here devcluster02)
```

Once we have the ssh keys created we are ready to start deploying nutch to all of the slave nodes.

(Note: this is a rather simple example of how to set up ssh without requiring a passphrase. There are other documents available which can help you with this if you have problems. It is important to test that the nutch user can ssh to all of the machines in your cluster so don't skip this stage)

Deploy Nutch to Single Machine

First we will deploy nutch to a single node, the master node, but operate it in distributed mode. This means that it will use the Hadoop filesystem instead of the local filesystem. We will start with a single node to make sure that everything is up and running and will then move on to adding the other slave nodes. All of the following should be done from a session started as the nutch user. We are going to setup nutch on the master node and then when we are ready we will copy the entire installation to the slave nodes.

First copy the files from the nutch build to the deploy directory using something like the following command:

```
cp -R /path/to/build/* /nutch/search
```

Then make sure that all of the shell scripts are in unix format and are executable.

```
dos2unix /nutch/search/bin/*.sh /nutch/search/bin/hadoop /nutch/search/bin/nutch
chmod 700 /nutch/search/bin/*.sh /nutch/search/bin/hadoop /nutch/search/bin/nutch
dos2unix /nutch/search/config/*.sh
chmod 700 /nutch/search/config/*.sh
```

When we were first trying to setup nutch we were getting bad interpreter and command not found errors because the scripts were in dos format on linux and not executable. Notice that we are doing both the bin and config directory. In the config directory there is a file called hadoop-env.sh that is called by other scripts.

There are a few scripts that you will need to be aware of. In the bin directory there is the nutch script, the hadoop script, the start-all.sh script and the stop-all.sh script. The nutch script is used to do things like start the nutch crawl. The hadoop script allows you it interact with the hadoop file system.
The start-all.sh script starts all of the servers on the master and slave nodes. The stop-all.sh. scrip stops all of the servers.

If you want to see options for nutch use the following command:

```
bin/nutch
```

Or if you want to see the options for hadoop use:

```
bin/hadoop
```

If you want to see options for Hadoop components such as the distributed filesystem then use the component name as input like below:

```
bin/hadoop dfs
```

There are also files that you need to be aware of. In the conf directory there are the nutch-default.xml, the nutch-site.xml, the hadoop-default.xml and the hadoop-site.xml. The nutch-default.xml file holds all of the default options for nutch, the hadoop-default.xml file does the same for hadoop. To override any of these options, we copy the properties to their respective *-site.xml files and change their values. Previously we had all the hadoop configuration in a file called hadoop-site.xml but recent versions have put hadoop related config in different files:

There is also a file named slaves inside the config directory. This is where we put the names of the slave nodes. Since we are running a slave data node on the same machine we are running the master node, we will also need the local computer in this slave list. Here is what the slaves file will look like to start.

```
localhost
```

It comes this way to start so you shouldn't have to make any changes. Later we will add all of the nodes to this file, one node per line.

Previously all the Hadoop configuration was in one file (hadoop-site.xml) but now we need to put roughly the same data in separate files. See http://hadoop.apache.org/common/docs/current/quickstart.html for more information. We are basically adding property entries inside the configuration tags...

conf/core-site.xml:

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://devcluster01:9000</value>
    <description>
       Where to find the Hadoop Filesystem through the network.
       Note 9000 is not the default port.
       (This is slightly changed from previous versions which didnt have "hdfs")
    </description>
  </property>
</configuration>
```

The fs.default.name property is used by nutch to determine the filesystem that it is going to use. Since we are using the hadoop filesystem we have to point this to the hadoop master or name node. In this case it is hdfs://devcluster01:9000 which is the server that houses the name node on our network.

The hadoop package really comes with two components. One is the distributed filesystem. Two is the mapreduce functionality. While the distibuted filesystem allows you to store and replicate files over many commodity machines, the mapreduce package allows you to easily perform parallel programming tasks.

conf/hdfs-site.xml:

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>

<property>
  <name>dfs.name.dir</name>
  <value>/nutch/filesystem/name</value>
</property>

<property>
  <name>dfs.data.dir</name>
  <value>/nutch/filesystem/data</value>
</property>

<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>

</configuration>
```

Note that the dfs-replication value of "1" means no duplication. This is only meaningful on a single machine test cluster. It should typically be 3 or more - but of course you must have at least that number of working nodes in your Hadoop cluster.

conf/mapred-site.xml:

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>

<property>
  <name>mapred.job.tracker</name>
  <value>devcluster01:9001</value>
  <description>
    The host and port that the MapReduce job tracker runs at. If
    "local", then jobs are run in-process as a single map and
    reduce task.
    Note 9001 is not the default port.
  </description>
</property>

<property>
  <name>mapred.map.tasks</name>
  <value>2</value>
  <description>
    define mapred.map tasks to be number of slave hosts
  </description>
</property>

<property>
  <name>mapred.reduce.tasks</name>
  <value>2</value>
  <description>
    define mapred.reduce tasks to be number of slave hosts
  </description>
</property>

<property>
  <name>mapred.system.dir</name>
  <value>/nutch/filesystem/mapreduce/system</value>
</property>

<property>
  <name>mapred.local.dir</name>
  <value>/nutch/filesystem/mapreduce/local</value>
</property>

</configuration>
```

The distributed file system has name nodes and data nodes. When a client wants to manipulate a file in the file system it contacts the name node which then tells it which data node to contact to get the file. The name node is the coordinator and stores what blocks (not really files but you can think of them as such for now) are on what computers and what needs to be replicated to different data nodes. The data nodes are just the workhorses. They store the actual files, serve them up on request, etc. So if you are running a name node and a data node on the same computer it is still communicating over sockets as if the data node was on a different computer.

I won't go into detail here about how mapreduce works, that is a topic for another tutorial and when I have learned it better myself I will write one, but simply put mapreduce breaks programming tasks into map operations (a -> b,c,d) and reduce operations (list -> a). Once a probelm has been broken down into map and reduce operations then multiple map operations and multiple reduce operations can be distributed to run on different servers in parallel. So instead of handing off a file to a filesystem node, we are handing off a processing operation to a node which then processes it and returns the result to the master node. The coordination server for mapreduce is called the mapreduce job tracker. Each node that performs processing has a daemon called a task tracker that runs and communicates with the mapreduce job tracker.

The nodes for both the filesystem and mapreduce communicate with their masters through a continuous heartbeat (like a ping) every 5-10 seconds or so. If the heartbeat stops then the master assumes the node is down and doesn't use it for future operations.

The mapred.job.tracker property specifies the master mapreduce tracker so I guess it is possible to have the name node and the mapreduce tracker on different computers. That is something I have not done yet.

The mapred.map.tasks and mapred.reduce.tasks properties tell how many tasks you want to run in parallel. This should be a multiple of the number of computers that you have. In our case since we are starting out with 1 computer we will have 2 map and 2 reduce tasks. Later we will increase these values as we add more nodes.

The dfs.name.dir property is the directory used by the name node to store tracking and coordination information for the data nodes.

The dfs.data.dir property is the directory used by the data nodes to store the actual filesystem data blocks. Remember that this is expected to be the same on every node.

The mapred.system.dir property is the directory that the mapreduce tracker uses to store its data. This is only on the tracker and not on the mapreduce hosts.

The mapred.local.dir property is the directory on the nodes that mapreduce uses to store its local data. I have found that mapreduce uses a huge amount of local space to perform its tasks (i.e. in the Gigabytes). That may just be how I have my servers configured though. I have also found that the intermediate files produced by mapreduce don't seem to get deleted when the task exits. Again that may be my configuration. This property is also expected to be the same on every node.

The dfs.replication property states how many servers a single file should be replicated to before it becomes available. Because we are using only a single server for right now we have this at 1. If you set this value higher than the number of data nodes that you have available then you will start seeing alot of (Zero targets found, forbidden1.size=1) type errors in the logs. We will increase this value as we add more nodes.

Before you start the hadoop server, make sure you format the distributed filesystem for the name node:

```
bin/hadoop namenode -format
```

And check the logs directory looking for errors.

Now that we have our hadoop configured and our slaves file configured it is time to start up hadoop on a single node and test that it is working properly. To start up all of the hadoop servers on the local machine (name node, data node, mapreduce tracker, job tracker) use the following command as the nutch user:

```
cd /nutch/search
bin/start-all.sh
```

To stop all of the servers you would use the following command:

```
bin/stop-all.sh
```

If everything has been setup correctly you should see output saying that the name node, data node, job tracker, and task tracker services have started. If this happens then we are ready to test out the filesystem. You can also take a look at the log files under /nutch/search/logs to see output from the different daemons services we just started.

You might want to look at http://localhost:50070/ with a web browser to confirm that the NameNode is up and running. (Replace localhost with devcluster01 or whatever you main host is called)

You can also look at http://localhost:50030/ to confirm that the JobTracker is up and running. (These ports seem to remain the same no matter that we entered "9000" and "9001" above.

To test the filesystem we are going to create a list of urls that we are going to use later for the crawl. Run the following commands:

```
cd /nutch/search
mkdir urlsdir
vi urlsdir/urllist.txt

http://lucene.apache.org
```

You should now have a urls/urllist.txt file with the one line pointing to the apache lucene site. Now we are going to add that directory to the filesystem. Later the nutch crawl will use this file as a list of urls to crawl. To add the urls directory to the filesystem run the following command:

```
cd /nutch/search
bin/hadoop dfs -put urlsdir urlsdir
```

You should see output stating that the directory was added to the filesystem. You can also confirm that the directory was added by using the ls command:

```
cd /nutch/search
bin/hadoop dfs -ls
```

Something interesting to note about the distributed filesystem is that it is user specific. If you store a directory urls under the filesystem with the nutch user, it is actually stored as /user/nutch/urls. What this means to us is that the user that does the crawl and stores it in the distributed filesystem must also be the user that starts the search, or no results will come back. You can try this yourself by logging in with a different user and runing the ls command as shown. It won't find the directories because is it looking under a different directory /user/username instead of /user/nutch.

If everything worked then you are good to add other nodes and start the crawl.

Deploy Nutch to Multiple Machines

The main point is to copy nutch-* (under $nutch_home/conf) and crawl-urlfilter.txt files to $hadoop_home/conf (all machines, including master and slaves) folder so that the hadoop cluster can pick up those configuration when startup. Otherwise nutch will complain with messages e.g. "0 records selected for fetching, exiting .. URLs to fetch - check your seed list and URL filters."

Once you have got the single node up and running we can copy the configuration to the other slave nodes and setup those slave nodes to be started out start script. First if you still have the servers running on the local node stop them with the stop-all script.

To copy the configuration to the other machines run the following command. If you have followed the configuration up to this point, things should go smoothly:

```
cd /nutch/search
scp -r /nutch/search/* nutch@computer:/nutch/search
```

Do this for every computer you want to use as a slave node. Then edit the slaves file, adding each slave node name to the file, one per line. You will also want to edit the hadoop-site.xml file and change the values for the map and reduce task numbers, making this a multiple of the number of machines you have. For our system which has 6 data nodes I put in 32 as the number of tasks. The replication property can also be changed at this time. A good starting value is something like 2 or 3. *(see Note at bottom about possibly having to clear filesystem of new datanodes). Once this is done you should be able to startup all of the nodes.

To start all of the nodes we use the exact same command as before:

```
cd /nutch/search
bin/start-all.sh
```

A command like 'bin/slaves.sh uptime' is a good way to test that things are configured correctly before attempting to call the start-all.sh script.

The first time all of the nodes are started there may be the ssh dialog asking to add the hosts to the known_hosts file. You will have to type in yes for each one and hit enter. The output may be a little wierd the first time but just keep typing yes and hitting enter if the dialogs keep appearing. You should see output showing all the servers starting on the local machine and the job tracker and data nodes servers starting on the slave nodes. Once this is complete we are ready to begin our crawl.

Performing a Nutch Crawl

Now that we have the the distributed file system up and running we can peform our nutch crawl. In this tutorial we are only going to crawl a single site. I am not as concerned with someone being able to learn the crawling aspect of nutch as I am with being able to setup the distributed filesystem and mapreduce.

To make sure we crawl only a single site we are going to edit crawl urlfilter file as set the filter to only pickup lucene.apache.org:

```
cd /nutch/search
vi conf/crawl-urlfilter.txt

change the line that reads:   +^http://([a-z0-9]*\.)*MY.DOMAIN.NAME/
to read:                      +^http://([a-z0-9]*\.)*apache.org/
```

We have already added our urls to the distributed filesystem and we have edited our urlfilter so now it is time to begin the crawl. To start the nutch crawl use the following command:

```
cd /nutch/search
bin/nutch crawl urlsdir -dir crawl -depth 3
```

We are using the nutch crawl command. The urlsdir is the urls directory that we added to the distributed filesystem. (I've called it "urlsdir" to make it clearer that it isn't merely the *file* containing urls). The "-dir crawl" is the output directory. This will also go to the distributed filesystem. The depth is 3 meaning it will only get 3 page links deep. There are other options you can specify, see the command documentation for those options.

You should see the crawl startup and see output for jobs running and map and reduce percentages. You can keep track of the jobs by pointing you browser to the master name node:

http://devcluster01:50070

and Mapreduce administration at

http://devcluster01:50030

You can also startup new terminals into the slave machine and tail the log files to see detailed output for that slave node. The crawl will probably take a while to complete. When it is done we are ready to do the search.

Testing the Crawl

You might want to try some of these commands before doing a search

```
bin/nutch readlinkdb crawl/linkdb -dump /tmp/linksdir
in nutch1.2  linkdb should be chaneged to crawldb bin/nutch readlinkdb crawl/crawldb -dump /tmp/linksdir
mkdir /nutch/search/output/
bin/hadoop dfs -copyToLocal /tmp/linksdir  /nutch/search/output/linksdir
less /nutch/search/output/linksdir/*
```

Or if we want to look at the whole thing as a text file we might try

```
bin/nutch readdb crawl/crawldb -dump /tmp/entiredump
bin/hadoop dfs -copyToLocal /tmp/entiredump  /nutch/search/output/entiredump
less /nutch/search/output/entiredump/*
```

Performing a Search

To perform a search on the index we just created within the distributed filesystem we need to do two things. First we need to pull the index to a local filesystem and second we need to setup and configure the nutch war file. Although technically possible, it is not advisable to do searching using the distributed filesystem.

The DFS is great for holding the results of the MapReduce processes including the completed index, but for searching it simply takes too long. In a production system you are going to want to create the indexes using the MapReduce system and store the result on the DFS. Then you are going to want to copy those indexes to a local filesystem for searching. If the indexes are too big (i.e. you have a 100 million page index), you are going to want to break the index up into multiple pieces (1-2 million pages each), copy the index pieces to local filesystems from the DFS and have multiple search servers read from those local index pieces. A full distributed search setup is the topic of another tutorial but for now realize that you don't want to search using DFS, you want to search using local filesystems.

Once the index has been created on the DFS you can use the hadoop copyToLocal command to move it to the local file system as such.

```
bin/hadoop dfs -copyToLocal crawl /d01/local/
```

Your crawl directory should have an index directory which should contain the actual index files. Later when working with Nutch and Hadoop if you have an indexes directory with folders such as part-xxxxx inside of it you can use the nutch merge command to merge segment indexes into a single index. The search website when pointed to local will look for a directory in which there is an index folder that contains merged index files or an indexes folder that contains segment indexes. This can be a tricky part because your search website can be working properly but if it doesn't find the indexes, all searches will return nothing.

If you setup the tomcat server as we stated earlier then you should have a tomcat installation under /nutch/tomcat and in the webapps directory you should have a folder called ROOT with the nutch war file unzipped inside of it. Now we just need to configure the application to use the distributed filesystem for searching. We do this by editing the hadoop-site.xml file under the WEB-INF/classes directory. Use the following commands:

```
cd /nutch/tomcat/webapps/ROOT/WEB-INF/classes
vi nutch-site.xml
```

Below is an template nutch-site.xml file:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

  <property>
    <name>fs.default.name</name>
    <value>local</value>
  </property>

  <property>
    <name>searcher.dir</name>
    <value>/d01/local/crawl</value>
  </property>

</configuration>
```

The fs.default.name property is now pointed locally for searching the local index. Understand that at this point we are not using the DFS or MapReduce to do the searching, all of it is on a local machine.

The searcher.dir directory is the directory where the index and resulting database are stored on the local filesystem. In our crawl command earlier we used the crawl directory which stored the results in "crawl" on the HDFS. Then we copied the crawl folder to our /d01/local directory on the local fileystem. So here we point this property to /d01/local/crawl. The directory which it points to should contain not just the index directory but also the linkdb, segments, etc. All of these different databases are used by the search. This is why we copied over the entire crawl directory and not just the index directory.

Once the nutch-site.xml file is edited then the application should be ready to go. You can start tomcat with the following command:

```
cd /nutch/tomcat
bin/startup.sh
```

Then point you browser to http://devcluster01:8080 (your search server) to see the Nutch search web application. If everything has been configured correctly then you should be able to enter queries and get results. If the website is working but you are getting no results it probably has to do with the index directory not being found. The searcher.dir property must be pointed to the parent of the index directory. That parent must also contain the segments, linkdb, and crawldb folders from the crawl. The index folder must be named index and contain merged segment indexes, meaning the index files are in the index directory and not in a directory below index named part-xxxx for example, or the index directory must be named indexes and contain segment indexes of the name part-xxxxx which hold the index files. I have had better luck with merged indexes than with segment indexes.

Distributed Searching

Although not really the topic of this tutorial, distributed searching needs to be addressed. In a production system, you would create your indexes and corresponding databases (i.e. crawldb) using the DFS and MapReduce, but you would search them using local filesystems on dedicated search servers for speed and to avoid network overhead.

Briefly here is how you would setup distributed searching. Inside of the tomcat WEB-INF/classes directory in the nutch-site.xml file you would point the searcher.dir property to a file that contains a search-servers.txt file. The search servers.txt file would look like this.

```
devcluster01 1234
devcluster01 5678
devcluster02 9101
```

Each line contains a machine name and port that represents a search server. This tells the website to connect to search servers on those machines at those ports.

On each of the search servers, since we are searching local directories, you would need to make sure that the filesystem in the nutch-site.xml file is pointing to local. One of the problems that I came across is that I was using the same nutch distribution to act as a slave node for DFS and MR as I was using to run the distributed search server. The problem with this was that when the distributed search server started up it was looking in the DFS for the files to read. It couldn't find them and I would get log messages saying x servers with 0 segments.

I found it easiest to create another nutch distribution in a separate folder. I would then start the distributed search server from this separate distribution. I just used the default nutch-site.xml and hadoop-site.xml files which have no configuration. This defaults the filesystem to local and the distributed search server is able to find the files it needs on the local box.

Whatever way you want to do it, if your index is on the local filesystem then the configuration needs to be pointed to use the local filesystem as show below. This is usually set in the hadoop-site.xml file.

```
<property>
 <name>fs.default.name</name>
  <value>local</value>
  <description>The name of the default file system.  Either the
  literal string "local" or a host:port for DFS.</description>
</property>
```

On each of the search servers you would use the startup the distributed search server by using the nutch server command like this:

```
bin/nutch server 1234 /d01/local/crawl
```

The arguments are the port to start the server on which must correspond with what you put into the search-servers.txt file and the local directory that is the parent of the index folder. Once the distributed search servers are started on each machine you can startup the website. Searching should then happen normally with the exception of search results being pulled from the distributed search server indexes. In the logs on the search website (usually catalina.out file), you should see messages telling you the number of servers and segments the website is attached to and searching. This will allow you to know if you have your setup correct.

There is no command to shutdown the distributed search server process, you will simply have to kill it by hand. The good news is that the website polls the servers in its search-servers.txt file to constantly check if they are up so you can shut down a single distributed search server, change out its index and bring it back up and the website will reconnect automatically. This way the entire search is never down at any one point in time, only specific parts of the index would be down.

In a production environment searching is the biggest cost both in machines and electricity. The reason is that once an index piece gets beyond about 2 million pages it takes too much time to read from the disk so you can have a 100 million page index on a single machine no matter how big the hard disk is. Fortunately using the distributed searching you can have multiple dedicated search servers each with their own piece of the index that are searched in parallel. This allow very large index system to be searched efficiently.

Doing the math, a 100 million page system would take about 50 dedicated search servers to serve 20+ queries per second. One way to get around having to have so many machines is by using multi-processor machine with multiple disks running multiple search servers each using a separate disk and index. Going down this route you can cut machine cost down by as much as 50% and electricity costs down by as much as 75%. A multi-disk machine can't handle the same number of queries per second as a dedicated single disk machine but the number of index pages it can handle is significantly greater so it averages out to be much more efficient.

Rsyncing Code to Slaves

Nutch and Hadoop provide the ability to rsync master changes to the slave nodes. This is optional though because it slows down the startup of the servers and because you might not want to have changed automatically synced to slave nodes.

If you do want this capability enabled then below I will show you how to configure your servers to rsync from the master. There are a couple of things you should know first. One, even though the slave nodes can rsync from the master you still have to copy the base installation over to the slave node the first time so that the scripts are available to rsync. This is the way we did it above so that shouldn't require any changeds Two the way the rsync happens is that the master node does an ssh into the slave node and calls bin/hadoop-daemon.sh. The script on the slave node then calls the rsync back to the master node. What this means is that you have to have a password-less login from each of the slave nodes to the master node. Before we setup password-less login from the master to the slaves, now we need to do the reverse. Three, if you have problems with the rsync options (I did and I had to change the options because I am running an older version of ssh), look in the bin/hadoop-daemon.sh script around line 82 for where it calls the rsync command.

So the first thing we need to do is setup the hadoop master variable in the conf/hadoop-env.sh file. The variable will need to look like this:

```
export HADOOP_MASTER=devcluster01:/nutch/search
```

This will need to be copied to all of the slave nodes like this:

```
scp /nutch/search/conf/hadoop-env.sh nutch@devcluster02:/nutch/search/conf/hadoop-env.sh
```

And finally you will need to log into each of the slave nodes, create a default ssh key for each machine and then copy it back to the master node where you will append it to the /nutch/home/.ssh/authorized_keys file. Here are the commands for each slave node, be sure to change the slavenodename when you copy the key file back to the master node so you don't overwrite files:

```
ssh -l nutch devcluster02
cd /nutch/home/.ssh

ssh-keygen -t rsa (Use empty responses for each prompt)
  Enter passphrase (empty for no passphrase):
  Enter same passphrase again:
  Your identification has been saved in /nutch/home/.ssh/id_rsa.
  Your public key has been saved in /nutch/home/.ssh/id_rsa.pub.
  The key fingerprint is:
  a6:5c:c3:eb:18:94:0b:06:a1:a6:29:58:fa:80:0a:bc nutch@localhost

scp id_rsa.pub nutch@devcluster01:/nutch/home/devcluster02.pub
```

Once you have done that for each of the slave nodes you can append the files to the authorized_keys file on the master node:

```
cd /nutch/home
cat devcluster*.pub >> .ssh/authorized_keys
```

With this setup whenever you run the bin/start-all.sh script files should be synced from the master node to each of the slave nodes.

Conclusion

I know this has been a lengthy tutorial but hopefully it has gotten you familiar with both nutch and hadoop. Both Nutch and Hadoop are complicated applications and setting them up as you have learned is not necessarily an easy task. I hope that this document has helped to make it easier for you.

If you have any comments or suggestions feel free to email them to me at nutch-dev@dragonflymc.com. If you have questions about Nutch or Hadoop they should be addressed to their respective mailing lists. Below are general resources that are helpful with operating and developing Nutch and Hadoop.

Updates

- I don't use rsync to sync code between the servers any more. Now I am using expect scripts and python scripts to manage and automate the system.
- I use distributed searching with 1-2 million pages per index piece. We now have servers with multiple processors and multiple disks (4 per machine) running multiple search servers (1 per disk) to decrease cost and power requirements. With this a single server holding 8 million pages can serve 10 queries a second constant.

Resources

Hadoop Quickstart: http://hadoop.apache.org/common/docs/current/quickstart.html

Google MapReduce Paper:
If you want to understand more about the MapReduce architecture used by Hadoop it is useful to read about the Google implementation.

http://labs.google.com/papers/mapreduce.html

Google File System Paper:
If you want to understand more about the Hadoop Distributed Filesystem architecture used by Hadoop it is useful to read about the Google Filesystem implementation.

http://labs.google.com/papers/gfs.html

Building Nutch - Open Source Search:
A useful paper co-authored by Doug Cutting about open source search and Nutch in paticular.

http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=144

Hadoop 0.1.2-dev API:

http://www.netlikon.de/docs/javadoc-hadoop-0.1/overview-summary.html

---

- - I, StephenHalsey, have used this tutorial and found it very useful, but when I tried to add additional datanodes I got error messages in the logs of those datanodes saying "2006-07-07 18:58:18,345 INFO org.apache.hadoop.dfs.DataNode: Exception: org.apache.hadoop.ipc.RemoteException: org.apache.hadoop.dfs.UnregisteredDatanodeException: Data node linux89-1:50010is attempting to report storage ID DS-1437847760. Expecting DS-1437847760.". I think this was because the hadoop/filesystem/data/storage file was the same on the new data nodes and they had the same data as the one that had been copied from the original. To get round this I turned everything off using bin/stop-all.sh on the name-node and deleted everything in the /filesystem directory on the new datanodes so they were clean and ran bin/start-all.sh on the namenode and then saw that the filesystem on the new datanodes had been created with new hadoop/filesystem/data/storage files and new directories and everything seemed to work fine from then on. This probably is not a problem if you do follow the above process without starting any datanodes because they will all be empty, but was for me because I put some data onto the dfs of the single datanode system before copying it all onto the new datanodes. I am not sure if I made some other error in following this process, but I have just added this note in case people who read this document experience the same problem. Well done for the tutorial by the way, very helpful. Steve.

- nice tutorial! I tried to set it up without having fresh boxes available, just for testing (nutch 0.8). I ran into a few problems. But I finally got it to work. Some gotchas:
  - use absolute paths for the DFS locations. Sounds strange that I used this, but I wanted to set up a single hadoop node on my Windows laptop, then extend on a Linux box. So relative path names would have come in handy, as they would be the same for both machines. Don't try that. Won't work. The DFS showed a ".." directory which disappeared when I switched to absolute paths.
  - I had problems getting DFS to run on Windows at all. I always ended up getting this exception: "Could not complete write to file e:/dev /nutch-0.8/filesystem/mapreduce/system/submit_2twsuj/.job.jar.crc by DFSClient_-1318439814 - seems nutch hasn't been tested much on Windows. So, use Linux.
  - don't use DFS on an NFS mount (this would be pretty stupid anyway, but just for testing, one might just set it up into an NFS homre directory). DFS uses locks, and NFS may be configured to not allow them.
  - When you first start up hadoop, there's a warning in the namenode log, "dfs.StateChange - DIR* FSDirectory.unprotectedDelete: failed to remove e:/dev/nutch-0.8/filesystem/mapreduce/.system.crc because it does not exist" - You can ignore that.

  - If you get errors like, "failed to create file [...] on client [foo] because target-length is 0, below MIN_REPLICATION (1)" this means a block could not be distributed. Most likely there is no datanode running, or the datanode has some severe problem (like the lock problem mentioned above).

- This tutorial worked well for me, however, I ran into a problem where my crawl wasn't working. Turned out, it was because I needed to set the user agent and other properties for the crawl. If anyone is reading this, and running into the same problem, look at the updated tutorial http://wiki.apache.org/nutch/Nutch0%2e9-Hadoop0%2e10-Tutorial?highlight=%28hadoop%29%7C%28tutorial%29

- By default Nutch will read only the first 100 links on a page. This will result in incomplete indexes when scanning file trees. So I set the "max outlinks per page" option to -1 in nutch-site.conf and got complete indexes.

```
<property>
  <name>db.max.outlinks.per.page</name>
  <value>-1</value>
  <description>The maximum number of outlinks that we'll process for a page.
  If this value is nonnegative (>=0), at most db.max.outlinks.per.page outlinks
  will be processed for a page; otherwise, all outlinks will be processed.
  </description>
</property>
```