Documentation for hanythingondemand

Release 20161021.01

Ghent University

Tue, 30 May 2017 07:44:46

Contents

1 Introductory topics **3**

1.1 What is hanythingondemand? . . . 3
1.2 Administration . . . 3
1.2.1 Prerequisites . . . 3
1.2.2 Torque . . . 4
1.2.3 Environment Modules . . . 4
1.3 Configuration . . . 5
1.3.1 Template parameters . . . 5
1.3.2 Service configs . . . 6
1.3.3 Autogenerated configuration . . . 6
1.3.4 Preview configuration . . . 6
1.4 Command line interface . . . 6
1.4.1 `hod` command . . . 7
1.4.2 `hod` subcommands . . . 8
1.5 Logging . . . 12
1.5.1 *hanythingondemand* logs . . . 12
1.5.2 Service logs . . . 12
1.6 Connecting to web user interfaces . . . 13
1.6.1 Setting up an SSH tunnel . . . 13
1.6.2 Browser SOCKS proxy configuration . . . 15
1.6.3 Ports for web user interfaces . . . 16
1.7 Spark . . . 16
1.8 Example use cases . . . 17
1.8.1 Common aspects . . . 17
1.8.2 Interactively using a Hadoop cluster . . . 18
1.8.3 Running a batch script on a Hadoop cluster . . . 20
1.8.4 Connecting to a Jupyter notebook running on an HOD cluster . . . 21

Welcome to the documentation of hanythingondemand (HOD, for short), a tool to set up and use an ad-hoc Hadoop cluster on HPC systems through a cluster resource manager like Torque/PBS.

https://github.com/hpcugent/hanythingondemand

This documentation is intended for hanythingondemand version 3.2.0, and was last rebuilt on Tue, 30 May 2017 07:44:46.

CHAPTER 1

Introductory topics

What is hanythingondemand?

hanythingondemand is a set of scripts to start a Hadoop cluster from within another resource management system (e.g., Torque/PBS). It allows traditional users of HPC systems to experiment with Hadoop or use it as a production setup if there is no dedicated setup available.

Administration

This is a page for Administrators who wish to offer *hanythingondemand* on their PBS/Torque cluster. hanythingondemand is a particularly tricky project as it integrates several pieces of tecchnology (Torque, MPI, Java, Hadoop, on Python) and as such we don't have an out of the box installation procedure yet.

Prerequisites

Here's an overview of the dependencies:

- A cluster using Torque
- environment-modules (used to test HOD) to manage the environment
- *Python* - 2.7.*
- Easybuild - we use Easybuild for installing software and hanythingondemand isn't tested without it.
- mpi4py
- vsc-base - Used for command line parsing.
- vsc-mympirun - Used for setting up the MPI job.
- pbs_python - Used for interacting with the PBS (aka Torque) server.
- Netifaces
- Netaddr
- *Java*
- Hadoop binaries

Torque

To use hanythingondemand you must be running a cluster that uses Torque as the resource manager.

Environment Modules

We use environment modules in conjunction with EasyBuild. You do not require environment-modules, however you will need to sort out all the paths for your users if you elect to not use it.

Easybuild

The following dependencies are installable using Easybuild. They should be pulled in when using the `eb hanythingondemand-${VERSION}.eb --robot` command:

mpi4py

EasyBuild scripts for mpi4py are available here

vsc-base

EasyBuild scripts for vsc-base are available here

vsc-mympirun

EasyBuild scripts for vsc-mympirun are available here

netifaces

EasyBuild scripts for netifaces are available here

netaddr

EasyBuild scripts for netaddr are available here

pbs_python

EasyBuild scripts for pbs_python are available here

Java

We use the Oracle JVM which isn't directly installable from EasyBuild since you need to register on the website to get the files.

Note: You'll need to manually download the JDK tarballs (requires registration) and seed them to EasyBuild.

Hadoop

We use Cloudera's Hadoop distribution and have tested with chd5.3.1.

Configuration

The configuration of hanythingondemand starts with the `hod.conf` file. This is an `ini` style configuration file with at least two sections: `Meta` and `Config`. Here is an example taken from the Hadoop 2.3 configs:

```
[Meta]
version=1

[Config]
modules=Hadoop/2.3.0-cdh5.0.0
master_env=HADOOP_HOME,EBROOTHADOOP,JAVA_HOME
services=resourcemanager.conf,nodemanager.conf,screen.conf
config_writer=hod.config.writer.hadoop_xml
workdir=/tmp
directories=$localworkdir/dfs/name,$localworkdir/dfs/data
autogen=hadoop
```

Here we have the Meta section with version set to 1. Version refers to the hanythingondemand configuration version. This is a placeholder in case we change the configurations around. That's all the `Meta` information is needed (for now). The following parameters are set in the `Config` section:

- `autogen` - Configuration files to autogenerate. This can be *hadoop*, *hadoop_on_lustre2*, or left blank. If it is set then hanythingondemand will create a basic configuration for you. This is particularly useful since it will calculate values for memory settings.You can then override any settings you feel necessary.
- `config_writer` - a reference to the python code that will output the configuration used by the services.
- `directories` - directories to create. If the service would fail without some directories being created, they should be entered here.
- `master_env` - environment variables to pass from the master node to the slave nodes. This is used because MPI slaves don't have an environment.
- `modules` - modules that must be loaded when the cluster begins.
- `services` - a list of service files containing start and stop script information.
- `workdir` - place where logs and temporary data is written. Configuration files will be copied here as well. `localworkdir` is a subdirectory of `workdir` and is useful for when `workdir` is on a shared file system.

Template parameters

There are some templating variables that can be entered into the configuration files. These use a dollar sign (`$`) prefix.

- `masterhostname` - hostname for the master node.
- `masterdataname` - hostname for the Infiniband interface of the master node
- `hostname` - hostname for the local node.
- `hostaddress` - ip for the local node.
- `dataname` - hostname for the Infiniband interface of the local node.
- `dataaddress` - ip for the Infiniband interface of the local node.
- `user` - user name of the person running the cluster.
- `pid` - process ID.
- `workdir` - workdir as defined.
- `localworkdir` - subdirectory of workdir qualified using the node name and a pid. This is used for keeping distinct per-node directories on a shared file system.

Service configs

Service configs have three sections: `Unit`, `Exec` and `Environment`. Here is an example:

```
[Unit]
Name=nodemanager
RunsOn=all

[Service]
ExecStart=$$EBROOTHADOOP/sbin/yarn-daemon.sh start nodemanager
ExecStop=$$EBROOTHADOOP/sbin/yarn-daemon.sh stop nodemanager

[Environment]
YARN_NICENESS=1 /usr/bin/ionice -c2 -n0 /usr/bin/hwloc-bind socket:0
HADOOP_CONF_DIR=$localworkdir/conf
YARN_LOG_DIR=$localworkdir/log
YARN_PID_DIR=$localworkdir/pid
```

- `Name` - name of the service.
- `RunsOn` - `(all|master|slave)`. Determines which nodes/group of nodes to run the service.
- `ExecStartPre` - script to run before starting the service. e.g. used in HDFS to run the `-format` script.
- `ExecStart` - script to start the service
- `ExecStop` - script to stop the service
- `Environment` - Environment variable definitions used for the service.

Autogenerated configuration

Autogenerating configurations is a powerful feature that lets you run services inside hanythingondemand on new clusters without having to hand calculate all the memory settings by hand.

For example,

- If your administrators installed a brand spanking new cluster with a large amount of memory available, you don't have to create a bunch of new configuration files to reflect the new system. It should all work seamlessly.
- You are holding a class and would like to allocate each student half a node - then they can use autogenerated settings along with `--rm-ppn=<half-the-number-of-cores>`

To autogenerate some configurations, set `autogen` setting to an appropriate value in the `Config` section.

Preview configuration

To preview the output configuration files that your `hod.conf` file would produce, one can use the `genconfig` command:

```
hod genconfig --hodconf=/path/to/hod.conf --workdir=/path/to/workdir
```

Here, `--workdir` is the output directory (which will be created if it doesn't yet exist) and `--hodconf` is the input configuration file.

Command line interface

This page provides an overview of the hanythingondemand command line interface.

Note: This only applies to hanythingondemand version 3 and newer; older versions provided a significantly different command line interface.

Contents

- *Command line interface*
  - *hod command*
    - *General hod command line options*
  - *hod subcommands*
    - *hod batch --script=<script-name>*
    - *hod clean*
    - *hod clone <dist-name> <destination>*
    - *hod connect <cluster-label>*
    - *hod create*
    - *hod destroy <cluster-label>*
    - *hod dists*
    - *hod genconfig*
    - *hod help-template*
    - *hod list*
    - *hod relabel <old-label> <new-label>*
    - *--job options for hod create / hod batch*

hod command

The main hanythingondemand command is a Python script named `hod`, and implements the top-level command line interface that discriminates between the various subcommands.

Running `hod` without arguments is equivalent to `hod --help`, and results in basic usage information being printed:

```
$ hod
hanythingondemand version 3.2.0 - Run services within an HPC cluster
usage: hod <subcommand> [subcommand options]
Available subcommands (one of these must be specified!):
    batch           Submit a job to spawn a cluster on a PBS job controller, run a␣
↪job script, and tear down the cluster when it's done
    clean           Remove stale cluster info.
    clone           Write hod configs to a directory for editing purposes.
    connect         Connect to a hod cluster.
    create          Submit a job to spawn a cluster on a PBS job controller
    destroy         Destroy an HOD cluster.
    dists           List the available distributions
    genconfig       Write hod configs to a directory for diagnostic purposes
    help-template   Print the values of the configuration templates based on the␣
↪current machine.
    list            List submitted/running clusters
    relabel         Change the label of an existing job.
```

General `hod` command line options

`hod [subcommand] --help`

Print usage information and supported subcommands along with a short help message for each of them, or usage information and available options for the specified subcommand.

`hod` subcommands

The `hod` command provides a number of subcommands, which correspond to different actions.

An overview of the available subcommands is available via `hod --help` (see *hod command*).

More details on a specific subcommand are available via `hod <subcommand> --help`.

Available subcommands:

- *hod batch –script=<script-name>*
- *hod clean*
- *hod clone <dist-name> <destination>*
- *hod connect <cluster-label>*
- *hod create*
- *hod destroy <cluster-label>*
- *hod dists*
- *hod genconfig*
- *hod help-template*
- *hod list*
- *hod relabel <old-label> <new-label>*

`hod batch --script=<script-name>`

Create a cluster and run the script. Upon completion of the script, the cluster will be stopped.

Next to `--script` (which is mandatory with `batch`), all configuration options supported for `create` are also supported for `batch`, see *Configuration options for hod create*. When used with `batch`, these options can also be specified via `$HOD_BATCH_*`.

Jobs that have completed will remain in the output of `hod list` with a job id of `<job-not-found>` until `hod clean` is run (see *hod clean*), or until the cluster is destroyed using `hod destroy` (see *hod destroy <cluster-label>*).

Note: `--hod-module`, `--workdir`, and either `--hodconf` or `--dist` **must** be specified.

`hod clean`

Remove cluster info directory for clusters that are no longer available, i.e. those marked with `<job-not-found>` in the output of `hod list`.

`hod clone <dist-name> <destination>`

Clone a dist for use editing purposes. If there is a provided dist that is almost what is required for some work, users can clone it and edit the files.

`hod connect <cluster-label>`

Connect to an existing hanythingondemand cluster, and set up the environment to use it.

This basically corresponds to logging in to the cluster head node using SSH and sourcing the cluster information script that was created for this cluster (`$HOME/.config/hod.d/<label>/env`).

`hod create`

Create a hanythingondemand cluster, with the specified label (optional) and cluster configuration file (required).

The configuration file can be a filepath, or one of the included cluster configuration files (see *hod dists*).

Jobs that have completed will remain in the output of `hod list` with a job id of `<job-not-found>` until `hod clean` is run (see *hod clean*), or until the cluster is destroyed using `hod destroy` (see *hod destroy <cluster-label>*).

Note: `--hod-module`, `--workdir`, and either `--hodconf` or `--dist` must be specified.

Configuration options for `hod create`

`hod create --hod-module <module name>`

must be specified

Specify the `hanythingondemand` module that must be loaded in the job that is submitted for the HOD cluster; can also be specified via `$HOD_CREATE_HOD_MODULE`.

`hod create --workdir <path>`

must be specified

Specify the top-level working directory to use; can also be specified via `$HOD_CREATE_WORKDIR`.

`hod create --hodconf <path>`

either `--dist` or this must be specified

Specify location of cluster configuration file; can also be specified via `$HOD_CREATE_HODCONF`.

`hod create --dist <dist>`

either `--hodconf` or this must be specified

Specify one of the included cluster configuration file to be used (see also *hod dists*); can also be specified via `$HOD_CREATE_DIST`.

hod create --label <label>

Specify label for this cluster. If not label is specified, the job ID will be used as a label; can also be specified via `$HOD_CREATE_LABEL`.

The label can be used to later connect to the cluster while it is running (see *hod connect <cluster-label>*).

hod create --modulepaths <paths>

Add additional locations for modules that need to be loaded (see *hod create –modules <module names>*).

Can also be specified via `$HOD_CREATE_MODULEPATHS`.

hod create --modules <module names>

Add modules to the dist so each node has access to them. If code submitted to the cluster requires a particular module, it should be added with this option. For example, if an IPython notebook plans to use Python modules on the worker kernels (or through Spark) they will need to be added here.

Can also be specified via `$HOD_CREATE_MODULES`.

hod create --job-*

The resources being requested for the job that is submitted can be controlled via the available `--job` options, see *–job options for hod create / hod batch*.

hod destroy <cluster-label>

Destroy the HOD cluster with the specified label.

This involves deleting the job, and removing the working directory and cluster info directory (`$HOME/.config/hod.d/<label>`) corresponding to this cluster, if they are still in place.

In case the cluster is currently *running*, confirmation will be requested.

hod dists

Print a list of available cluster configurations (‘*distributions*‘), along with the list of modules that correspond to each of them.

See for example *Available distributions*.

hod genconfig

Generate hanythingondemand cluster configuration files to the working directory for diagnostic purposes.

The working directory can be specified using `--workdir` or via `$HOD_GENCONFIG_WORKDIR`.

hod help-template

Print the values for the configuration templates based on the current machine.

`hod list`

Print a list of existing clusters, and their state (‘`queued`‘ or ‘`running`‘).

Jobs that have completed running will remain in the list with `<job-not-found>` until `hod clean` is run (see *hod clean*), or until the HOD cluster is destroyed using `hod destroy` (see *hod destroy <cluster-label>*).

`hod relabel <old-label> <new-label>`

Change the label for a hod cluster that is queued or running.

`--job` options for `hod create` / `hod batch`

The `create` and `batch` subcommands accept the following options to specify requested job resources.

These can also be specified via `$HOD_BATCH_JOB_*` (for `hod batch`) or `$HOD_CREATE_JOB_*` (for `hod create`).

`--job-mail/-m <string>`

Send a mail when the cluster has started (`b` for ‘*begin*‘), stopped (`e` for ‘*ended*‘) or got aborted (`a`).

For example, using `-m a` will result in receiving a mail whn the cluster has started running.

`--job-mailothers/-M <main addresses>`

List of other mail adresses to send mails to.

`--job-name/-N <name>`

Specify the name for the job that will be submitted.

`--job-nodes/-n <int>`

The number of (full) workernodes to request for the job being submitted (default: 1).

`--job-ppn <int>`

The number of cores per workernode to request; by default: `-1`, i.e. full workernodes (request all available cores).
`--job-reservation <reservation ID>`

Reservation ID to submit job into (equivalent with using `-W x=FLAGS:ADVRES:<reservation ID>` with `qsub`).

`--job-queue/-q <int>`

Name of job queue to submit to (default: none specified).

`--job-walltime/-l <int>`

Number of hours of walltime to request (default: 48).

Logging

If your job didn't work as expected, you'll need to check the logs.

It's important to realise that both *hanythingondemand* itself and the services it is running (e.g. Hadoop) produce logs.

Which logs you should be diving into depends on the information you are looking for or the kind of problems you run into.

Contents

- *Logging*
  - hanythingondemand *logs*
  - *Service logs*

hanythingondemand logs

For *hanythingondemand* itself, there are three places to consider:

1. When submitting your job to start the cluster, hanythingondemand logs to your terminal session. The potential errors here are usually:
   - PBS isn't running or isn't accessible. If so, contact your administrators.
   - Your environment is broken. For example, if you're using a Python version for a cluster that doesn't work on the login node.
2. If PBS is accessible and tries to run the job but it failed to start properly (e.g. due to a problem with MPI) you will see errors in `Hanythingondemand.e${PBS_JOBID}`. This will be in the directory from where you ran the job.
3. When PBS starts your job, it will start logging to `hod.output.$(hostname).$(pid)`. If your service configuration files have problems (e.g. typos in the commands, bad paths, etc) then the error will be here. For example if a service failed to start you will see a message in the logs saying: `Problem occured with cmd`.

Service logs

Hadoop logs

By default, the log files for a Hadoop cluster will be in `$HOD_LOCALWORKDIR/log`, where the `$HOD_LOCALWORKDIR` is an environment variable set by `hod connect`.

Expanded, this is in the workdir of the HOD cluster as follows: `$workdir/$PBS_JOBID/${USER}.${HOSTNAME}.${PID}/log`

One of the advantages of having the log files on a parallel file system is that one no longer needs to use special tools for log aggregation (Flume, Logstash, Logly, etc) since all the logs for all the nodes are in a single directory structure.

Hadoop logs have two components:

1. *Service logs*: These are in `$HOD_LOCALWORKDIR/log`. Examples are: `yarn-username-resourcemanager-node.domain.out`, `yarn-username-nodemanager-node.domain.out`.
2. *Container logs*: Each piece of Hadoop work takes place in a container. Output from your program will appear in these files. These are organized by application/container/stderr and stdout. For example:

```
$HOD_LOCALWORKDIR/log/userlogs/application_1430748293929_0001/container_
↪1430748293929_0001_01_000003/stdout
```

IPython logs

IPython logs to stdout and stderr. These are sent by hanythingondemand to `$HOD_LOCALWORKDIR/log/pyspark.stdout` and `$HOD_LOCALWORKDIR/log/pyspark.stderr`

`hod batch` logs

Logs for your script running under `hod batch` are found in your `$PBS_O_WORKDIR` in: `<script-name>.o<$PBS_JOBID>` and `<script-name>.e<$PBS_JOBID>`.

If you want to watch the progress of your job while it's running, it's advisable to write your script so that it pipes output to the `tee` command.

Connecting to web user interfaces

To connect to web user interfaces (UIs) that are available for a running HOD cluster, you need to follow these steps:

1. Set up an SSH tunnel to the head node of your HOD cluster (see *Setting up an SSH tunnel*)
2. Configure your browser to use the SSH tunnel as a SOCKS proxy (see *Browser SOCKS proxy configuration*)
3. Point your browser to `http://localhost:<port>` (see *Ports for web user interfaces*)

Contents

- *Connecting to web user interfaces*
  - *Setting up an SSH tunnel*
    - *Determine hostname of head node of HOD cluster*
    - *Configuring your SSH client to use an SSH tunnel*
    - *Starting the SSH tunnel*
  - *Browser SOCKS proxy configuration*
  - *Ports for web user interfaces*
    - *Ports for Hadoop web user interface (defaults)*
    - *Ports for Spark web services*
    - *Ports for IPython web services*

Setting up an SSH tunnel

To connect to a web UI available on your running *hanythingondemand* cluster, you most likely need to set up an SSH tunnel first.

Typically, the HOD cluster is running on a workernode of an HPC cluster that is only accessible via the HPC login nodes. To connect to a web UI however, we need *direct* access. This can be achieved by tunneling via SSH over the login nodes.

To set up an SSH tunnel, follow these steps:

1. Determine hostname of the head node of your HOD cluster (see *Determine hostname of head node of HOD cluster*)
2. Configure your SSH client (see *Configuring your SSH client to use an SSH tunnel*)
3. Start the SSH tunnel (see *Starting the SSH tunnel*)

Determine hostname of head node of HOD cluster

The first step is to figure out which workernode is the *head* node of your HOD cluster, using `hod list`.

For example:

```
$ hod list
Cluster label   Job ID                              State   Hosts
example         123456.master15.delcatty.gent.vsc R       node2001.delcatty.gent.
↪vsc
```

So, in this example, `node2001.delcatty.gent.vsc` is the *fully qualified domain name (FQDN)* of the head node of our HOD cluster.

Configuring your SSH client to use an SSH tunnel

See the sections below how to configure your SSH client.

Configuring SSH in Mac OS X or Linux

To configure SSH to connect to a particular workernode using an SSH tunnel, you need to add a couple of lines to your `$HOME/.ssh/config` file.

For example, to configure SSH that it should tunnel via the HPC login node `login.hpc.ugent.be` for all FQDNs that start with `node` and end with `.gent.vsc`, using `vsc40000` as a user name, the following lines should be added:

```
Host login.hpc.ugent.be hpc
    Hostname login.hpc.ugent.be
    User vsc40000
    IdentityFile ~/.ssh/id_rsa

Host node*.gent.vsc
    ProxyCommand ssh -q login.hpc.ugent.be 'exec nc -w 21600s %h %p'
    User vsc40000
    IdentityFile ~/.ssh/id_rsa
```

Make sure to point it to your ssh private key (`~/.ssh/id_rsa` in this example). If you only have a single private key, this line can be dropped. Adjust `vsc40000` to your own VSC username.

Configuring PuTTY in Windows

configuring_putty_on_windows is more involved than Linux or OS X so it has its own page.

Starting the SSH tunnel

To start the SSH tunnel, simply set up an SSH connection to that head node of your HOD cluster, while specifying a *local port* that can be used to set up a SOCKS proxy to that workernode.

You can choose a port number yourself, but stick to numbers **higher than 1024** (lower ports are priveledged ports, and thus require adminstration rights).

We will use port number `10000` (*ten thousand*) as an example below (and you should be able to use it too).

Starting the SSH tunnel on Mac OS X or Linux

On OS X or Linux, just SSH to the FQDN of the head node of the HOD cluster, and specify the local port you want to use for your SOCKS proxy via the `-D` option of the SSH command.

For example, to connect to `node2001.delcatty.gent.vsc` using port `10000`:

```
$ ssh -D 10000 node2001.delcatty.gent.vsc
$ hostname
node2001.delcatty.os
```

Note: Starting the SSH tunnel will only work if you have an HOD cluster running on the specified workernode. If not, you may see the connection 'hang' rather than fail. To cancel to connection attempt, use Ctrl-C.

Note: When first connecting to a workernode, you will see a request to accept the RSA key fingerprint for that workernode, as shown below. If you are confident you are connecting to the right workernode, enter '*yes*':

```
The authenticity of host 'node2001.delcatty.gent.vsc (<no hostip for proxy command>
↪)' can't be established.
RSA key fingerprint is 00:11:22:33:44:55:66:77:88:99:aa:bb:ee:dd:ee:ff.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'node2001.delcatty.gent.vsc' (RSA) to the list of known␣
↪hosts.
```

Starting the SSH tunnel using PuTTy in Windows

With your saved session configured, open the proxy session.

Browser SOCKS proxy configuration

To access the web user interface(s) of your running HOD cluster, you need to configure your browser to use the SSH tunnel as a proxy.

Basically, you need to:

- define `localhost` (i.e., your own system) as a SOCKS proxy in your browser, using the port that you used when setting up the SSH tunnel (e.g., `10000`)
- make sure that the proxy will also be used when entering `https://localhost:<port>` as a URL in your browser
- enter `https://localhost:<port>` as a URL in your browser, with `<port>` the port number for the web UI you want to connect to (see *Ports for web user interfaces*)

The pages linked below provide a detailed walkthrough with screenshots on how to configure some commonly used browsers:

- Firefox (Windows, OS X, Linux)
- Chrome, Safari (OS X)
- Chrome (Windows)

Note: Keep in mind that using the proxy will only work while you have access to the workernode for which the SSH tunnel was set up, i.e. while the HOD cluster is running, and while you are able to connect to the HPC infrastructure.

To reset your browser configuration back to normal, simply disable the proxy in your browser configuration.

Ports for web user interfaces

Once you have set up an SSH tunnel (see *Setting up an SSH tunnel*) and have configured your browsers to use it as a SOCKS proxy (see *Browser SOCKS proxy configuration*), you can connect to the web user interfaces available in your running HOD cluster via:

```
http://localhost:<port>
```

The port number to use depends on the particular web user interface you want to connect to, see below.

Note: The command `netstat -tulpn` may be helpful in figuring out the ports being used by the running services.

Ports for Hadoop web user interface (defaults)

- `50030`: Hadoop job tracker
- `50060`: Hadoop task tracker
- `50070`: HFDS name node
- `50075`: HDFS data nodes
- `50090`: HDFS secondary name node
- `50105`: HDFS backup/checkpoint node

(see also http://blog.cloudera.com/blog/2009/08/hadoop-default-ports-quick-reference)

Ports for Spark web services

- `4040`: information about running Spark application

Note: If multiple Spark applications (SparkContexts) are running, their web UI will be available via successive ports beginning with 4040 (4041, 4042, etc).

(see also https://spark.apache.org/docs/latest/monitoring.html)

Ports for IPython web services

- `8888`: IPython notebook

Spark

Many traditional HPC users who are interested in *Hadoop* are also very interested in Spark. With the claimed performance improvements, it's easy to see why.

Spark works with Yarn out of the box so there's nothing special that needs to happen with Hanythingondemand. Just use `spark-submit --master=yarn-cluster` with all the other arguments to spark-submit, and it works.

Example use cases

A couple of example use cases are described below.

We assume that the `hod` command is readily available in the environment; if it is not by default, maybe you should load a module first: see which `hod` or `hanythingondemand` modules are available via `module avail`, and load one of them using `module load`.

To check, just run `hod` without arguments, which should produce basic usage information (see *hod command*).

Contents

- *Example use cases*
  - *Common aspects*
  - *Interactively using a Hadoop cluster*
  - *Running a batch script on a Hadoop cluster*
  - *Connecting to a Jupyter notebook running on an HOD cluster*

Common aspects

Configuring HOD

You can/should configure HOD by defining the HOD work directory and specifying which module should be loaded in the HOD job being submitted (see also *Configuration options for hod create*).

To configure `hod batch`, you can set the following environment variables:

```
$ env | grep HOD_BATCH
HOD_BATCH_HOD_MODULE=hanythingondemand/3.0.0-intel-2015b-Python-2.7.10
HOD_BATCH_WORKDIR=$VSC_SCRATCH/hod
```

Likewise, for `hod create`:

```
$ env | grep HOD_CREATE
HOD_CREATE_HOD_MODULE=hanythingondemand/3.0.0-intel-2015b-Python-2.7.10
HOD_CREATE_WORKDIR=$VSC_SCRATCH/hod
```

The examples below will assume that this configuration is in place already.

Available distributions

To get an overview of readily available HOD distributions, to select a value to specify to `--dist`, use `hod dists` (slightly trimmed output):

```
$ hod dists
* HBase-1.0.2
    modules: HBase/1.0.2, Hadoop/2.6.0-cdh5.4.5-native
...
* Hadoop-2.6.0-cdh5.4.5-native
    modules: Hadoop/2.6.0-cdh5.4.5-native
```

```
...
* Jupyter-notebook-5.1.0
    modules: Hadoop/2.6.0-cdh5.8.0-native, Spark/2.0.0, IPython/5.1.0-intel-2016b-
↪Python-2.7.12, matplotlib/1.5.1-intel-2016b-Python-2.7.12
```

Interactively using a Hadoop cluster

To interactively use an HOD cluster, you should

1. create an HOD cluster, using `hod create`
2. connect to it once it is up and running, using `hod connect`
3. execute your commands

See the example below for more details; basic usage information for `hod create` is available at *hod create*.

Using `screen`

To interactively start commands that may require some time to finish, we strongly recommended starting a so-called *screen* session after connecting to the HOD cluster.

Basic usage:

- to start a screen session, simply the `screen` command; to specify a name for the session, use `screen -S <name>`
- to get an overview of running screen sessions, use `screen -ls`
- to detach from a screen session, with the option to later reattach to it, us the `Ctrl-A-D` key combination.
- to *end* a screen session, simply type `exit` (no reattaching possible later!)
- to reconnect to a screen session, use `screen -r <name>`; or simply use `screen -r` if there's only one running screen session

More information about `screen` is available at http://www.gnu.org/software/screen/manual/screen.html.

Example: Hadoop WordCount

In the example below, we create a Hadoop HOD cluster, connect to it, and run the standard WordCount example Hadoop job.

- create a Hadoop HOD cluster labelled `hod_hadoop`, requesting 1 workernode:

```
$ hod create --dist Hadoop-2.5.0-cdh5.3.1-native --label hod_hadoop --job-
↪nodes 1

Submitting HOD cluster with label 'hod_hadoop'...
Job submitted: Jobid 12345.master15.delcatty.gent.vsc state Q ehosts
```

- check the status of the HOD cluster (‘`Q`‘ for queued, ‘`R`‘ for running):

```
$ hod list

Cluster label       Job ID                                State       Hosts
hod_hadoop          12345.master15.delcatty.gent.vsc        Q

...

$ hod list
```

```
Cluster label       Job ID                                  State       Hosts
hod_hadoop          12345.master15.delcatty.gent.vsc        R       node2001.
↪delcatty.gent.vsc
```

- connect to the running HOD cluster:

```
$ hod connect hod_hadoop

Connecting to HOD cluster with label 'hod_hadoop'...
Job ID found: 12345.master15.delcatty.gent.vsc
HOD cluster 'hod_hadoop' @ job ID 12345.master15.delcatty.gent.vsc appears to␣
↪be running...
Setting up SSH connection to node2001.delcatty.gent.vsc...
Welcome to your hanythingondemand cluster (label: hod_hadoop)

Relevant environment variables:
HADOOP_CONF_DIR=/user/scratch/gent/vsc400/vsc40000/hod/hod/12345.master15.
↪delcatty.gent.vsc/vsc40000.node2001.delcatty.os.26323/conf
HADOOP_HOME=/apps/gent/CO7/haswell-ib/software/Hadoop/2.5.0-cdh5.3.1-native/
↪share/hadoop/mapreduce
HOD_LOCALWORKDIR=/user/scratch/gent/vsc400/vsc40000/hod/hod/12345.master15.
↪delcatty.gent.vsc/vsc40000.node2001.delcatty.os.26323

List of loaded modules:
Currently Loaded Modulefiles:
  1) cluster/delcatty(default)        2) Java/1.7.0_76                  3)␣
↪Hadoop/2.5.0-cdh5.3.1-native
```

- run Hadoop WordCount example
  - copy-paste the source code for the WordCount example from https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v1.0 into a file named `WordCount.java` in your home directory:

    ```
    $ nano $HOME/WordCount.java
    ```

  - change to local work directory of this cluster:

    ```
    $ cd $HOD_LOCALWORKDIR
    ```

  - download example input file for wordcount:

    ```
    $ curl http://www.gutenberg.org/files/98/98.txt -o tale-of-two-cities.txt
    ```

  - build `WordCount.jar` (*note:* assumes that `$HOME/WordCount.java` is available):

    ```
    $ cp $HOME/WordCount.java .
    $ javac -classpath $(hadoop classpath) WordCount.java
    $ jar cf WordCount.jar WordCount*.class
    ```

  - run `WordCount` Hadoop example:

    ```
    $ hadoop jar WordCount.jar WordCount tale-of-two-cities.txt wordcount.out
    # (output omitted)
    ```

  - query result:

    ```
    $ grep ^city wordcount.out/part-r-00000
    city    20
    city,   9
    city.   5
    ```

Running a batch script on a Hadoop cluster

Since running a pre-defined set of commands is a common pattern, HOD also supports an alternative to creating an HOD cluster and using it interactively.

Via `hod batch`, a script can be provided that should be executed on an HOD cluster. In this mode, HOD will:

- start an HOD cluster with the specified configuration (working directory, HOD distribution, etc.)
- execute the provided script
- automatically destroy the cluster once the script has finished running

This alleviates the need to wait until a cluster effectively starts running and entering the commands interactively.

See also the example below; basic usage information for `hod batch` is available at *hod batch –script=<script-name>*.

Example: Hadoop WordCount

The classic Hadoop WordCount can be run using the following script (`wordcount.sh`) on an HOD cluster:

```
#!/bin/bash

# move to (local) the local working directory of HOD cluster on which this script␣
↪is run
cd $HOD_LOCALWORKDIR

# download example input file for wordcount
curl http://www.gutenberg.org/files/98/98.txt -o tale-of-two-cities.txt

# build WordCount.jar
# note: assumes that ``$HOME/WordCount.java`` is available, copy-paste the source␣
↪code from
# https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-
↪client-core/MapReduceTutorial.html#Example:_WordCount_v1.0
cp $HOME/WordCount.java .
javac -classpath $(hadoop classpath) WordCount.java
jar cf WordCount.jar WordCount*.class

# run WordCount Hadoop example
hadoop jar WordCount.jar WordCount tale-of-two-cities.txt wordcount.out

# copy results
cp -a wordcount.out $PBS_O_WORKDIR/wordcount_${PBS_JOBID}.out
```

Note: No modules need to be loaded in order to make sure the required software is available (i.e., Java, Hadoop). Setting up the working environment in which the job will be run is done right after starting the HOD cluster.

To check which modules are/will be available, you can use `module list` in the script you supply to `hod batch` or check the details of the HOD distribution you use via *hod clone <dist-name> <destination>*.

To run this script on a Hadoop cluster, we can submit it via `hod batch`:

```
$ hod batch --dist Hadoop-2.5.0-cdh5.3.1-native --script $PWD/wordcount.sh --label␣
↪wordcount
Submitting HOD cluster with label 'wordcount'...
Job submitted: Jobid 12345.master15.delcatty.gent.vsc state Q ehosts

$ hod list
Cluster label       Job ID                              State       Hosts
wordcount           12345.master15.delcatty.gent.vsc        R       node2001.
↪delcatty.gent.vsc
```

Once the script is finished, the HOD cluster will destroy itself, and the job running it will end:

```
$ hod list
Cluster label       Job ID                                State                Hosts
wordcount           12345.master15.delcatty.gent.vsc        <job-not-found> <none>
```

Hence, the results should be available (see the `cp` at the end of the submitted script):

```
$ ls $HOME/HOD_wordcount.12345.master15.delcatty.gent.vsc
total 416
-rw-r--r-- 1 example  example  210041 Oct 22 13:34 part-r-00000
-rw-r--r-- 1 example  example       0 Oct 22 13:34 _SUCCESS

$ grep ^city $HOME/HOD_wordcount.12345.master15.delcatty.gent.vsc/part-r-00000
city        20
city,       9
city.       5
```

Note: To get an email when the HOD cluster is started/stopped, use the `-m` option, see *–job-mail/-m <string>*.

Connecting to a Jupyter notebook running on an HOD cluster

Running a Jupyter notebook on an HOD cluster is as simple as creating an HOD cluster using the appropriate distribution, and then connecting to the Jupyter notebook over an SSH tunnel.

For example:

- create HOD cluster using a Jupyter HOD distribution:

```
$ hod create --dist Jupyter-notebook-5.1.0 --label ipython_example
Submitting HOD cluster with label 'ipython_example'...
Job submitted: Jobid 12345.master15.delcatty.gent.vsc state Q ehosts
```

- determine head node of HOD cluster:

```
$ hod list
Cluster label       Job ID                                State       Hosts
ipython_example 12345.master15.delcatty.gent.vsc    R       node2001.delcatty.
↪gent.vsc
```

- connect to Jupyter notebook by pointing your web browser to http://localhost:8888, using a SOCKS proxy over an SSH tunnel to the head node `node2001.delcatty.gent.vsc` (see *Connecting to web user interfaces* for detailed information)