Data Transfer

Copying data from ARCHER to the RDF

1. Aims

The aim of this exercise is to experiment with the various methods of copying data between ARCHER and the RDF, and compare their performance. Although the details of the timings will be specific to the ARCHER service, the general trends should hold true for any pair of systems.

2. Introduction

We will investigate three aspects:

1. the relative performance of `cp` and `rsync`;
2. collecting files together into a single archive;
3. whether or not compression is used.

The datasets we will experiment with are:

1. the source for the sharpen exercise (already downloaded);
2. a collection of images from a medical scan (see `headsq.tar`);
3. single large test files containing zeroes or junk data;
4. a collection containing many large test files (created using `generate.sh`).

You can create large files for testing purposes using the (somewhat archaic) Unix command `dd`, e.g. to create a 1MB file called mb1.dat:

```
user@archer:~> dd if=/dev/zero of=mb1.dat bs=$[1024*1024] count=1
```

This creates a file very quickly, but note that because it is filled with zeroes you may get unrepresentative results if you use compression. You can create a file containing pseudo-random bytes:

```
user@archer:~> dd if=/dev/zero of=mb1rand.dat bs=1 count=$[1024*1024]
```

but this takes a lot longer. If you want to create very large files that are not filled with zeros, you can combine smaller files, e.g.

```
user@archer:~> cat mb1rand.dat mb1rand.dat > mb2rand.dat
user@archer:~> cat mb2rand.dat mb2rand.dat > mb4rand.dat
user@archer:~> cat mb4rand.dat mb4rand.dat > mb8rand.dat
```

etc. etc.

To time any Unix operation (e.g. copying, compressing, unpacking, ...) you should just use the `time` command, e.g.

```
user@archer:~> time dd if=/dev/random of=m1zero.dat bs=1
count=$[1024*1024]

1048576+0 records in
1048576+0 records out
1048576 bytes (1.0 MB) copied, 3.52683 s, 297 kB/s

real   0m3.530s
user   0m0.044s
sys    0m3.472s
```

Performance can vary significantly on any shared system, particularly if you are doing IO, so it is worth repeating any measurements several times to check that you are getting representative timings.

3. Exercises

You should compare the performance of the following:

1. Copying a large file: `cp largefile.dat /general/y14/y14/guestXX/`
2. Recursively copying a directory using:
   `cp -r directory /general/y14/y14/guestXX/directory`
3. Copying the same directory by first creating an archive using `tar`.
4. Compressing/uncompressing the archive before/after copying.

Remember to include the time taken for archiving/compressing etc. when assessing the overall performance.

Repeat the above tests with the following changes:

1. Using `rsync` rather than `cp`. Note that you must include the option `-a` to ensure that only the new files are transferred; the verbose option `-v` is useful to see exactly what `rsync` is doing (e.g. which files are copied).
2. Use `cpio` instead of `tar`.
3. Use `zip` (with and without compression) instead of `tar`.

Extra exercises

1. All the datasets used above may be rather small (either in terms file size of number of files) to show any dramatic effect. We have therefore provided you with a utility program `generate.sh` to enable you easily to create a large number of files of different sizes. See the inbuilt help for instructions:

   ```
   user@archer:~> bash ./generate.sh
   Generate a number of random files in the working directory.
   Usage: generate.sh [OPTIONS].

     -h, --help            displays this message
     -n, --number-of-files number of files to be generated
     -l, --lower-limit     smallest possible file size in bytes
     -u, --upper-limit     largest possible file size in bytes

   Example:
     generate.sh -n 500 -l 524288 -u 2097152
   Creates 500 random files of between 512 KiB and 2 MiB.
   ```

   The default behaviour (no options) is 500 random files of between 512 KiB and 2 MiB.

2. Rather than performing the copying, archiving etc. on the login nodes, use the post-processing nodes. You will have to submit a PBS batch script to the serial queue on ARCHER as described in the slides.

3. We will cover network transfers later on, but if you want to you can look at the performance of ssh between the systems; ssh uses encryption which introduces some overheads. Points to note are:

   a. you will need to have ssh keys installed so you are not prompted for your password every time;
   b. you should use one of the DTNs as the endpoint for the transfer.