Access to Big Data in Bioinformatics

Andrew van Rooyen
University of Cape Town

ABSTRACT

1. BACKGROUND

Data sets are a big part of bioinformatics, and have introduced many new challenges with the rise of next generation sequencing. Sequencing technologies like SOLiD provide much higher data output at a cheaper cost [Shendure and Ji 2008], which is good news for research, but troubling for data storage, transfer and access. In fact, the cost of storing a byte has been more expensive than sequencing a base pair since before 2010 [Baker 2010].

This makes it difficult for researchers in different locations to manipulate and run processes on the data, because it will be stored in only one location. These files could be tens of gigabytes in size [Deorowicz and Grabowski 2011], depending on context.

1.1 Data storage

Generally, once the sequencing machine has generated the raw information on each base pair, this data will be stored in a data warehouse. Storing this information for long periods of time requires the data to be structured efficiently in order to save space, and allow it to be transferred efficiently. There has been a lot of work on how to structure this data. There are a plethora of file formats whose efficiency depends on the kind of data which needs to be stored. Two of the most popular formats are FASTQ, which stores aggregated reads along with the quality of each base pair [Cock et al. 2010], and BAM, the binary, compressed version of the Sequence Alignment Map (SAM) format [SAMTools 2015].

1.2 Data transfer

When researchers require specific information for their projects, they need to be able to access the data warehouse and transfer whichever sequences they need. Luckily, these locations are often connected massive data pipes like National Research and Education Networks (NRENs). Unfortunately, standard protocols like FTP and SSH were never designed for use on high-throughput networks, and alternate protocols need to be used to avoid bottlenecking.

There are some proprietary transfer protocols which are widely used in practice. For example, the fasp protocol by the US based company AsperaSoft. Based on UDP, the protocol eliminates the latency issues seen with TCP, and provides bandwidth up to 10 gigabits per second to transfer data [Beloslyudtsev 2014].

1.3 Alternate models

There have been some attempts to do data processing remotely, and there has been an explorative push towards cloud solutions from Amazon, Google etc. Unfortunately, even though these cloud data centres have plenty of cheap storage, there are very significant drawbacks.

Because the sequencing happens in labs, researchers need to upload their raw data to the cloud data centres every time they run a new experiment. This leads back to the original problem, as researchers resort to mailing hard drives [Baker 2010]. There are also security, privacy and ethical concerns with outsourcing this processing power to other companies, as sequenced DNA data is often highly sensitive information [Marx 2013].

2. DESIGN AND IMPLEMENTATION

2.1 Design Aims

The system will provide mechanisms to compare how the following protocols handle file transfers. This information will be used to determine which one is best to use when dealing with bioinformatics data on an educational network.

- GridFTP
- FTP
- HPN-SSH
- SCP

The protocols to test were specified by the proposer of the project. They are also the most popular protocols in the field, perhaps with the exception of *fasp* by AsperaSoft [Beloslyudtsev 2014], which is non-free. Because of the popularity of these protocols over all others, it is assumed that one of them will be the ideal protocol.

2.2 Approach

Transfers using each of the 4 protocols above will be run while the network traffic is logged. This information will

ACM ISBN 000000.
DOI: 000000

then be analysed.

A simple python script will be sufficient to run the transfers, as most of the work will be done by calling a subprocess for each specific protocol. The analysis will also be done using python as there are a vast number of analytical and visualisation tools available.

The testing environment will be kept as stable as possible during tests, and multiple tests will be run at different times of the day.

First, the scripts which run the transfers will be written so that tests can be run and results saved. The analysis suite will be written later, and will accept the previously-saved results as input.

The testbed will be constructed using git for version control. The repository is available at https://github.com/wraithy/bigbinf.

With regards to GridFTP, the 'lite' version will be used. This means that authentication is done over ssh as opposed a previously-configured certificate authority. This does not make any difference to the file transfer itself, but it does prevent unnecessary configuration of the testbed which can be quite complex in the case of 'full' GridFTP.

Once the ideal protocol has been decided on, it will be made available as an endpoint to the microcloud system, so that users can retrieve their results in an optimal way.

2.3 Evaluation

The system will log packets on the network interface used for the transfer. This allows for analysis which goes deeper then simply logging the transfer speed and file size reported by the program.
Metrics that will be investigated include

- Average speed, max speed
- Consistency
- Total transferred bytes and overhead

2.4 Implementation

All testing will be done on an Ubuntu 14.04 system, with the following packages installed

- python 2.7
- globus-gass-copy-progs 8.6
- globus-gridftp-server-progs 6.38
- tcpdump
- openssh-client
- openssh-server

However, the testbed should work on any Unix system as long as python, tcpdump and the correct programs for each protocol are installed.

2.4.1 File transfer

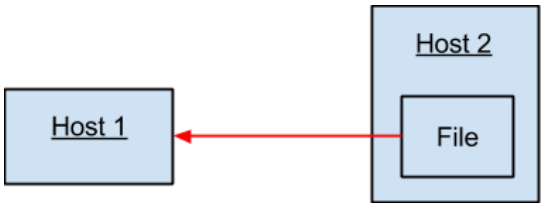


Figure 1: Host 1 copying a file.

A copy is initiated from Host 1. A file from Host 2 is then transferred to Host 1 using the particular protocol.

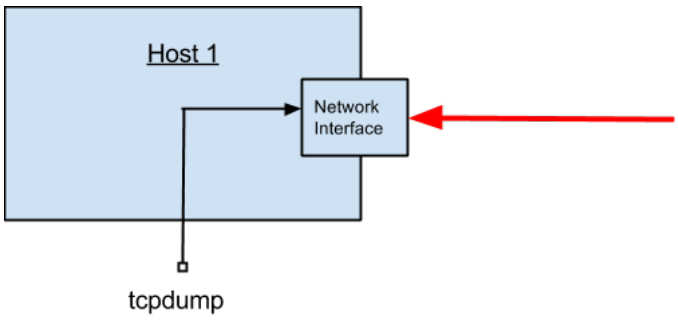


Figure 2: The network interface capturing traffic.

The 'tcpdump' program (www.tcpdump.org) comes with most Unix systems. It watches a network interface (e.g eth0) and logs information about packets which pass through. The program is run while the copy is in progress, and the output is filtered to include only packets sent between Host 1 and Host 2.

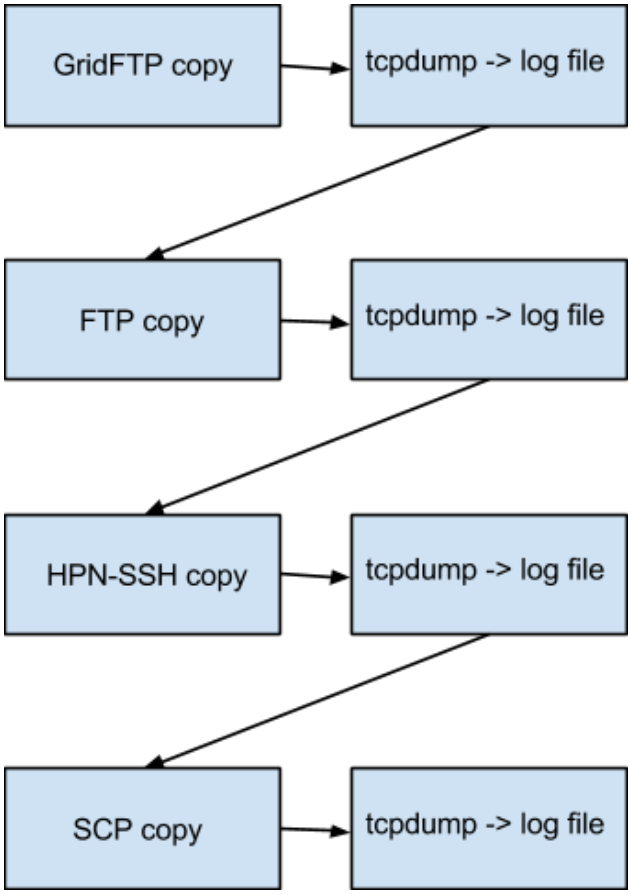
Note that despite the name, tcpdump can also capture UDP packets, which is relevant for some of the protocols.

2.4.2 Transfer logging

The python script for running the file transfers has been written to accept

- the name of the network interface
- the remote hostname (Host 2)
- the path of the file on Host 2
- a local path to copy the file to

It then resolves the IPs of each host, and for each protocol, runs a copy in isolation. It spawns a tcpdump subprocess, and lets it run for precisely as long as the copy runs. The tcpdump program is started with filters, so that only traffic between the two hosts is captured. It then saves the output in a file.



Doing it this way allows for a much more controlled environment. Because the tcpdump is only capturing while the copy is running, no other packets will be included in the logs. Also, the copies are run programmatically and consecutively. Following copies are not started until both the tcpdump and protocol processes have been closed, and the log file has been written. This means that they are all run in an identical (within reason) environment, but at the same time do not interfere with each other.

This test process is run multiple times for statistical reasons, generating multiple log files.

2.4.3 Analysis of dumps

A separate python file reads in the log files and parses them. Operations can then be run by looking at the time of each packet, and the size of its payload. This allows analysis of

- overhead, because *all* transferred is logged, rather than just the file size
- speed (data/time)
- consistency
- total time
- total size

This data can then be aggregated over multiple log files, and graphed using the matplotlib python library.

More info is needed here, and will be filled in once I have completed the analysis scripts.

3. REFERENCES

[Baker 2010] Monya Baker. 2010. Next-generation sequencing: adjusting to data overload. *nature methods* 7, 7 (2010), 495–499.

[Beloslyudtsev 2014] Dima Beloslyudtsev. 2014. Aspera transfer guide. (2014).

[Cock et al. 2010] Peter JA Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. 2010. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research* 38, 6 (2010), 1767–1771.

[Deorowicz and Grabowski 2011] Sebastian Deorowicz and Szymon Grabowski. 2011. Compression of DNA sequence reads in FASTQ format. *Bioinformatics* 27, 6 (2011), 860–862.

[Marx 2013] Vivien Marx. 2013. Biology: The big challenges of big data. *Nature* 498, 7453 (2013), 255–260.

[SAMTools 2015] SAMTools. 2015. Sequence Alignment/Map Format Specification. https://samtools.github.io/hts-specs/SAMv1.pdf. (2015). Accessed: 2015-04-27.

[Shendure and Ji 2008] Jay Shendure and Hanlee Ji. 2008. Next-generation DNA sequencing. *Nature biotechnology* 26, 10 (2008), 1135–1145.