Step-by-Step guide for downloading very large datasets to a supercomputer using the SRA Toolkit

Jacob Heldenbrand,Yingxue Ren,Yan Asmann,Liudmila S. Mainzer

Abstract

Citation: Jacob Heldenbrand,Yingxue Ren,Yan Asmann,Liudmila S. Mainzer Step-by-Step guide for downloading very large datasets to a supercomputer using the SRA Toolkit . **protocols.io**

dx.doi.org/10.17504/protocols.io.kb6csre

Published: 17 Oct 2017

Protocol

Introduction

Step 1.

The SRA Toolkit is a complex piece of software that can be difficult to navigate, as the documentation is extensive and error messages are not always able to provide clarity when failures occur. In an effort to maximize the Toolkit's utility, we have devised a protocol for downloading thousands of SRA files and converting them into FASTQ files in a reasonable amount of time. To make the process as simple as possible, our protocol anticipates and adjusts for likely errors. While this guide has a limited lifespan in the face of regular updates to the Toolkit, we hope that our instructions will be of help to the community, as they summarize a lot of disjoint information already floating around on the Web.

Tools

Step 2.

- prefetch—For downloading the SRA files themselves from NCBI
- vdb-config—Must use this to configure the toolkit and specify the location of the dbGaP private key
- sra-validate—Tool that performs a checksum on SRA to ensure transfer of data was successful
- fastq-dump—For converting the SRA files into the FASTQ format for easy use
- Anisimov Launcher—Blue Waters tool that launches multiple jobs in parallel
- Aspera—Download tool

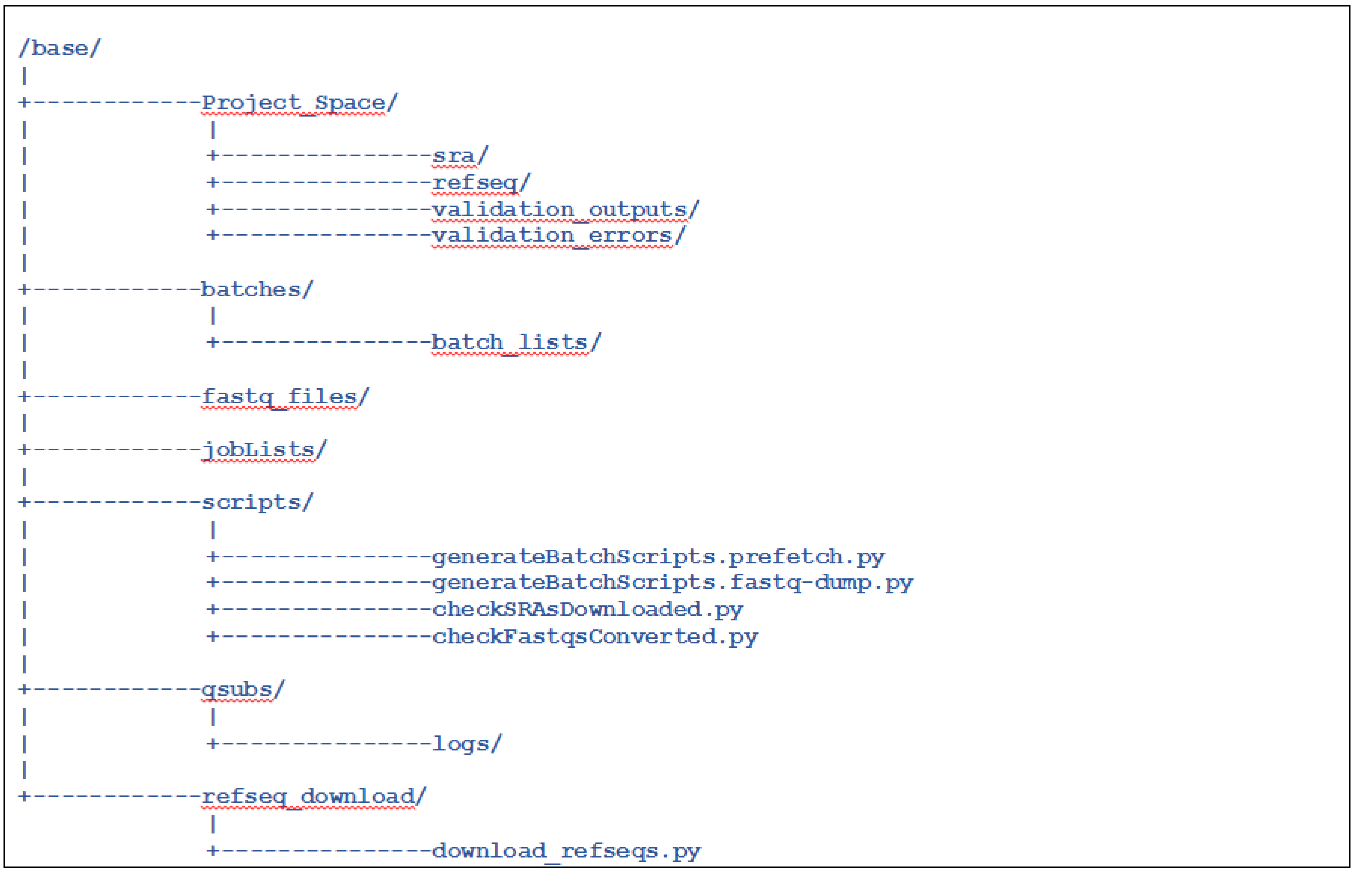
Note1: This protocol assumes you are downloading dbGaP data. If not, skip the private key configuration steps.

Note2: We designed this Guide for downloads to Blue Waters. With small adjustments, it should be applicable to other clusters.

Assumed File Structure

Step 3.

The explanations and scripts below assume the following file structure. If it is modified, the scripts must be altered as well.



```
/base/
|
+------------Project_Space/
|             |
|             +---------------sra/
|             +---------------refseq/
|             +---------------validation_outputs/
|             +---------------validation_errors/
|
+------------batches/
|             |
|             +---------------batch_lists/
|
+------------fastq_files/
|
+------------jobLists/
|
+------------scripts/
|             |
|             +---------------generateBatchScripts.prefetch.py
|             +---------------generateBatchScripts.fastq-dump.py
|             +---------------checkSRAsDownloaded.py
|             +---------------checkFastqsConverted.py
|
+------------qsubs/
|             |
|             +---------------logs/
|
+------------refseq_download/
              |
              +---------------download_refseqs.py
```

Preparation

Step 4.

The download procedure is normally a two-step process: first grab the SRA files from the repository, then convert SRA files to FASTQ on the cluster. A few preparatory steps will help avoid bottlenecks.

vdb-config

Run the following command to execute vdb-config (located within the SRA toolkit bin folder). This may require X11 forwarding (ssh –X flag on login to cluster).

```
./vdb-config –i
```

This opens a GUI where the location of the dbGaP project space can be configured. Set this to Project_Space. If downloading dbGaP data, specify the repository key location.

Make sure this is done before downloading the refseq data below, as the tool will not allow you to point to a directory in which the sra and refseq subdirectories are not empty.

refseq download

To convert an SRA file to the FASTQ format, fastq-dump must normally download reference data stored in a refseq database at NCBI. However, this creates a bottleneck when trying to scale up conversions of many files, as the reference data end up being downloaded repeatedly for every file batch.

To circumvent this bottleneck, we manually downloaded all the reference files located at https://ftp.ncbi.nlm.nih.gov/sra/refseq/. While this is a large download of 40GB, it only needs to be done once. Furthermore, the SRA Toolkit is configured to download any missing reference files if it cannot find them later during the SRA to FASTQ conversion stage. Thus, if new reference files are added to the repository between your bulk reference download and the actual data conversion, you should still get correct results when running fastq-dump.

To download the reference files, copy the contents of https://ftp.ncbi.nlm.nih.gov/sra/refseq/ into an Excel sheet, grab the names of each file, and put them in a file named /base/refseq_download/list_all_refseqs.txt.

Use the following bash script as a wrapper to call the python script that downloads the reference files. Wrap the bash script in a qsub and submit it to a compute node. Using ten download processes in parallel by breaking up /base/refseq_download/list_all_refseqs.txt into 10 batches will increase efficiency.

```
/base/refseq_download/download_parallel_wrapper.sh

#!/bin/bash

python base/refseq_download/download_refseqs_parallel.py 0 &
python base/refseq_download/download_refseqs_parallel.py 1 &
python base/refseq_download/download_refseqs_parallel.py 2 &
python base/refseq_download/download_refseqs_parallel.py 3 &
python base/refseq_download/download_refseqs_parallel.py 4 &
python base/refseq_download/download_refseqs_parallel.py 5 &
python base/refseq_download/download_refseqs_parallel.py 6 &
python base/refseq_download/download_refseqs_parallel.py 7 &
python base/refseq_download/download_refseqs_parallel.py 8 &
python base/refseq_download/download_refseqs_parallel.py 9 &
wait
```

```
/base/refseq_download/download_refseqs_parallel.py

import sys
import subprocess
```

```
N = int(sys.argv[1])

filenames = []

with open('/base/refseq_download/list_all_refseqs.txt') as F:
        for line in F:
                name = line.strip()
                filenames.append(name)

# Start at position N and go to the end in 10 step intervals
for i in filenames[N::10]:
        subprocess.check_call("/.aspera/connect/bin/ascp \
        -i /.aspera/connect/etc/asperaweb_id_dsa.openssh -k 1 -T -l800m \
        anonftp@ftp.ncbi.nlm.nih.gov:/sra/refseq/{0} \
        /base/Project_Space/refseq/".format(i), shell=True)
```

Data Download: SRA files

Step 5.

Step 1: Create Batch List

For each batch, create a text file in /base/batches/batch_lists/. For this protocol, we will refer to the current batch being downloaded as batchX. Therefore, create a file like the following:

```
/base/batches/batch_list/batchX.txt
```

SRR123
SRR234
SRR345
SRR456
SRR567

... and so on.

Note: Using this procedure we downloaded 10,000 SRA files broken up into batches of 1,200 SRAs.

Step 2: Create JobList and prefetch bash scripts for the Anisimov Launcher

This step is designed to bundle individual single-threaded download tasks into an MPI job that can run across multiple nodes. This increases queue priority, and facilitates efficient use of nodes on clusters that espouse node exclusivity (no more than one user per node). The launcher is a simple MPI wrapper, which takes in a list of all the individual tasks (JobList.txt) and places them on the available cores within the multi-node qsub reservation on the cluster. If you give it more tasks than cores, then it will start the first batch of tasks on the available cores, and keep starting new ones as the tasks complete and cores become available. For each ID in this batch, use the generateBatchScripts.prefetch.py script to automatically create a bash script like the following. This is your “Anisimov task” for this batch:

```
#!/bin/bash

# If downloading dbGaP data, prefetch must be called from within the project
space folder
cd /base/Project_Space

# Download the SRA file
/path/to/sra-toolkit/bin/prefetch -L debug -t fasp -v -v <SAMPLE_ID>
```

The generateBatchScripts.prefetch.py will also construct the JobList.txt file that lists the names and locations of these scripts in the following format:

```
/base/batches/batchX/SRR123 SRR123.sh
/base/batches/batchX/SRR234 SRR234.sh
/base/batches/batchX/SRR345 SRR345.sh
/base/batches/batchX/SRR456 SRR456.sh
/base/batches/batchX/SRR567 SRR567.sh
```

... and so on.

Use this Python script to generate both the shell scripts for each sample and the jobList file:

```
/base/scripts/generateBatchScripts.prefetch.py (modify with paths to suit
your needs)

#!/usr/bin/python

import os
import os.path
import sys

### GLOBAL VARIABLES

sraListFile = sys.argv[1]
batchName = sys.argv[2]

batchFullPath = "base/batches/" + batchName

SRA_list = []

### FUNCTION DEFINITIONS

def createScripts(SRA_ID):
    subDirName = batchFullPath + "/" + SRA_ID

    # Create the subdirectory within the batch directory
    if (not os.path.isdir(subDirName)):
        os.mkdir(subDirName)
    # Create the shell script file
    shellFile = open(subDirName + "/" + SRA_ID + ".sh", "w")
    # Write to the file
    shellFile.write("#!/bin/bash\n\n")

    shellFile.write("cd base/Project_Space\n\n")

    shellFile.write("# Download the SRA file\n")
    shellFile.write("/path/to/sra-toolkit/bin/prefetch -L debug -t fasp -v -v
" \
                    + SRA_ID + "\n\n")
    shellFile.close()

def makeJobListFile():
    jobListFile = open("base/jobLists/" + batchName + "_JobList.txt", "w")
    for i in SRA_list:
        # Write the jobList for the Anisimov launcher
        # Something like "/base/batches/batch1/SRR123 SRR123.sh"
        jobListFile.write(batchFullPath + "/" + i + " " + i + ".sh\n")
    jobListFile.close()

### IMPLEMENTATION

# Get the list of SRA IDs
with open(sraListFile) as F:
    for line in F:
        SRA_list.append(line.strip())

# If the batch directory does not exist, create it
if (not os.path.isdir(batchFullPath)):
    os.mkdir(batchFullPath)
# Create the subdirectories and shell scripts
for i in SRA_list:
    createScripts(i)

makeJobListFile()
```

There are two arguments passed to this script at runtime: the location of the batchX.txt file created earlier and the name of the batch. It can be invoked with the following command:

```
python generateBatchScripts.prefetch.py /base/batches/batch_lists/batchX.txt
batchX
```

This will generate the jobList.txt file and the necessary bash scripts and put them in their own directory within the batches folder:

```
/base/batches/batchX/SRR123/SRR123.sh
/base/batches/batchX/SRR234/SRR234.sh
/base/batches/batchX/SRR345/SRR345.sh
```

... and so on.

Step 3: Create the prefetch qsub script

To use the Anisimov Launcher to schedule jobs, construct a qsub script. These are stored in the /base/qsubs directory. Our tests suggest that 15 samples can be downloaded on a node simultaneously, as long as they are spaced out over the cores (on a cray system we supply the “-d 2” flag to aprun).

```
#nodes = batch_size/15
Aprun’s -n flag = #nodes * 16
Aprun’s -N flag = 16
```

The qsub script should look something like the example qsub below, which assumes a batch of 1,200 samples.

```
/base/qsubs/batchX_prefetch.qsub

#!/bin/bash
#PBS -N sra_X
#PBS -l walltime=1:15:00
#PBS -l nodes=80:ppn=32
#PBS -A groupid
#PBS -q normal

aprun -n 1280 -N 16 -ss -d 2 anisimov/scheduler/scheduler.x \
  /base/jobLists/batchX_JobList.txt /bin/bash -noexit &>
/base/qsubs/logs/batchX_prefetch.log
```

This script will launch all the bash scripts (15/node). Both stdout and stderr will be piped to /base/qsubs/logs/batchX_prefetch.log.

Step 4: Running the prefetch qsub script

Unfortunately, as prefetch runs, some of the SRA downloads will fail. To prevent a single failure from killing the Anisimov Job and the other downloads occurring in parallel, the -noexit flag is used (see the box above). However, this means the download may eventually reach a point at which all the SRAs have finished downloading, but the job just sits without making progress. This is just a consequence of the Anisimov Launcher code design.

To prevent this from wasting resources, monitor the size of the /base/Project_Space/sra folder during the download using the following command:

```
ls –l /base/Project_Space/sra | head
```

If the size of the sra/ folder does not appear to grow for five minutes or so, go ahead and kill the job (yes, it is a hack at this point, and one could automate it if desired):

```
qdel JobID
```

As the SRAs are downloaded, temporary files are generated in the sra/ folder. If those files are present the next time you attempt to download this ID, the download will fail. To prevent this from happening, delete the .tmp and .lock files with the following commands:

```
cd /base/Project_Space/sra

rm *.tmp.aspera-ckpt
rm *.tmp.partial
rm *.lock
rm *.tmp

rm *.vdbcache.cache
rm *.vdbcache
```

Step 5: Determine which SRA IDs did not finish downloading

After removing the .tmp and .lock files, run the following script:

```
/bash/scripts/checkSRAsDownloaded.py

import sys
import glob

batch_list = sys.argv[1]

# List of IDs
batch_IDs = []

with open(batch_list) as F:
    for line in F:
        batch_IDs.append(line.strip())

IDs_found = []

for f in glob.glob("/base/Project_Space/sra/*"):
    split_string = f.split("/")
    ID = split_string[-1].split('.')[0]
    IDs_found.append(ID)

# Remove redundant
IDs_found = list(set(IDs_found))

count_missing = 0

for i in batch_IDs:
    if i not in IDs_found:
        print(i)
        count_missing += 1
```

```
print("\nIDs that are missing")
print(count_missing)
```

This script grabs the IDs in the batch file and checks to see whether each SRA file is found in the sra/ folder. Invoke with the following command:

```
python /base/scripts/checkSRAsDownloaded.py
/base/batches/batch_lists/batchX.txt
```

Any SRA IDs that are not present in the sra/ folder will be printed out, as well as the total number that were not downloaded. Copy these IDs and put them in a new batch.txt file in /base/batches/batch_lists. We found it effective to name this new file batchX.1, then name the next iteration batchX.2, and so on. Repeat the downloading steps until all SRA IDs are accounted for.

Note: On each iteration, reduce the resources requested in each qsub script so that resources are not wasted.

Data Conversion: SRA to fastq.gz

Step 6.

Because the refseq reference data ares already downloaded, it will be easy to convert the SRA files to fastq.gz files.

However, first, it makes sense to check that the SRA files are intact using sra-validate. Call both

sra-validate and fastq-dump in the same shell script, as shown in the following example script:

```
#!/bin/bash

cd /base/Project_Space

/path/to/sra-toolkit/bin/vdb-validate <SAMPLE_ID>.sra &>   \
    /base/Project_Space/validation_outputs/batchX/<SAMPLE_ID>.validation_out

if grep -q 'err'
/base/Project_Space/validation_outputs/batchX/<SAMPLE_ID>.validation_out;
then
        echo 'Verification of <SAMPLE_ID>.sra failed'
        cp
/base/Project_Space/validation_outputs/batchX/<SAMPLE_ID>.validation_out  \
/projects/sciteam/baib/InputData_DoNotTouch/dbGaP-13335/validation_failures/b
atch5
else
        echo 'No errors found in <SAMPLE_ID>.sra'
        # Convert the SRA into fastq
        /path/to/sra-toolkit/bin/fastq-dump -v --gzip --split-files \
               -O /base/fastq_files/batchX
```

```
/base/Project_Space/sra/<SAMPLE_ID>.sra
fi
```

This script will run sra-validate and store its output in the validation_out. If an error is found, the output is copied into the validation_error/ folder and fastq-dump is not run. Otherwise, fastq-dump runs as expected.

These Anisimov launcher scripts are generated in the same way that the prefetch scripts were generated earlier, with a python script that generates the jobList.txt file and the shell scripts for each sample (next page).

```
/base/scripts/generateBatchScripts.fastq-dump.py (modify with paths to suit
your needs)

#!/usr/bin/python

import os
import os.path
import sys



### GLOBAL VARIABLES

sraListFile = sys.argv[1]
batchName = sys.argv[2]

batchFullPath = "/base/batches/" + batchName

SRA_list = []

### FUNCTION DEFINITIONS

def createScripts(SRA_ID):
    subDirName = batchFullPath + "/" + SRA_ID

    # Create the subdirectory within the batch directory
    if (not os.path.isdir(subDirName)):
        os.mkdir(subDirName)
    # Create the shell script file
    shellFile = open(subDirName + "/" + SRA_ID + ".sh", "w")

    # Write to the file
    shellFile.write("#!/bin/bash\n\n")

    shellFile.write("cd /base/Project_Space\n\n")

    shellFile.write("/path/to/sra-toolkit/bin/vdb-validate " + SRA_ID + ".sra
&> \
                     /base/Project_Space/validation_outputs/" + batchName +
"/" +  SRA_ID \
                     + ".validation_out\n\n")

    shellFile.write("if grep -q 'err'
/base/Project_Space/validation_outputs/" \
                     + batchName + "/" + SRA_ID + ".validation_out; then\n")
    shellFile.write("\techo 'Verification of " + SRA_ID + ".sra failed'\n")
    shellFile.write("\tcp /base/Project_Space/validation_outputs/" +
batchName + "/" \
                      + SRA_ID + ".validation_out
/base/Project_Space/validation_failures/" \
                      + batchName + "\n")
    shellFile.write("else\n")
    shellFile.write("\techo 'No errors found in " + SRA_ID + ".sra'\n")
    shellFile.write("\t# Convert the SRA into fastq\n")
    shellFile.write("\t/path/to/sra-toolkit/bin/fastq-dump -v --gzip --split-
files \
                      -O /base/fastq_files/" + batchName + "
/base/Project_Space/sra/" \
                      + SRA_ID + ".sra\n")
    shellFile.write("fi\n")

    shellFile.close()
def makeJobListFile():
    jobListFile = open("/base/jobLists/" + batchName + "_JobList.txt", "w")
    for i in SRA_list:
        # Write the jobList for the Anisimov launcher
        # Something like "/base/batches/batch1/SRR123 SRR123.sh"
        jobListFile.write(batchFullPath + "/" + i + " " + i + ".sh\n")
    jobListFile.close()


### IMPLEMENTATION

# Get the list of SRA IDs
with open(sraListFile) as F:
    for line in F:
        SRA_list.append(line.strip())

# If the batch directory does not exist, create it
if (not os.path.isdir(batchFullPath)):
    os.mkdir(batchFullPath)



# Create the subdirectories and shell scripts
for i in SRA_list:
    createScripts(i)

makeJobListFile()

try:# Create directory in the validation folders
    os.mkdir("/base/Project_Space/validation_outputs/" + batchName)
    os.mkdir("/base/Project_Space/validation_failures/" + batchName)
except:
    pass
```

This script is invoked in the same way that the prefetch script generator was:

```
cd /base/scripts
python generateBatchScripts.fastq-dump.py ../batches/batch_lists/batchX.txt
batchX
```

Note: This script will overwrite the jobList and the batch scripts generated from the prefetch generator for batchX. However, all the files will have already been downloaded.

Create and run the fastq-dump qsub script

The SRA to FASTQ conversion itself typically proceeds without error. Although the batch was downloaded in iterations, the whole batch can generally be converted in one step.

The qsub looks something like the following, assuming a batch size of 1,200 samples:

```
/base/qsubs/batchX_fastq-dump.qsub

#!/bin/bash
#PBS -N sra_X
#PBS -l walltime=6:00:00
#PBS -l nodes=80:ppn=32
#PBS -A groupid
#PBS -q normal

aprun -n 1280 -N 16 -ss -d 2 anisimov/scheduler/scheduler.x
/base/jobLists/batchX_JobList.txt /bin/bash -noexit &>
/base/qsubs/logs/batchX_fastq-dump.log
```

This script will launch all the bash scripts (15/node). Both stdout and stderr will be piped to /base/qsubs/logs/batchX_fastq-dump.log.

After this script completes, run the following script to verify that all the fastq files are present:

```
/base/scripts/checkFastqsConverted.py

#!/usr/bin python
"""
This script checks to see how many IDs in a given list are found within the
sra folder
It prints those that are not present

"""
import sys
import glob

batch_list = sys.argv[1]
fastq_batch = sys.argv[2]

# List of IDs
batch_IDs = []

with open(batch_list) as F:
    for line in F:
        batch_IDs.append(line.strip())

fastqs_found = []

for f in glob.glob(fastq_batch + "/*"):
    split_line = f.split("/")
    fastqs_found.append(split_line[-1])

count_missing = 0



for i in batch_IDs:
    fq1 = i + "_1.fastq.gz"
    fq2 = i + "_2.fastq.gz"
    if fq1 not in fastqs_found:
        print fq1
        count_missing += 1
    else:
        # Remove it from the list, so if any IDs are left in the end, those
IDs should not be in this directory
        fastqs_found.remove(fq1)
    if fq2 not in fastqs_found:
        print fq2
        count_missing += 1
    else:
        fastqs_found.remove(fq2)

print("\nFastq files that are missing")
print(count_missing)

print("\nThese IDs were found but shouldn't be here")
print(fastqs_found)
```

Invoke with the following command:

```
python /base/scripts/checkFastqsConverted.py
/base/batches/batch_lists/batchX.txt /base/fastq_files/batchX
```

At this point, the download and conversion are complete. If any fastq files are absent, inspect the validation_error files to find out why. Re-download the SRA files if necessary.

You now have a complete set of FASTQ files from NCBI. We hope you found this protocol useful.

Acknowledgements

Step 7.

We are grateful to Ms. Katherine Kendig for editorial help with this guide. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.