# FRESCO: Open Source Data Repository for Computational Usage and Failures

From Purdue's Central Computing Clusters

Supported by NSF Grant No. CNS-1548114

## Introduction

FRESCO is a repository of performance data for scientific code execution jobs submitted to Purdue University's Conte campus cluster. Data in the repository can be used to identify failed jobs and analyze reasons for failure by studying the performance parameters during the job's execution on individual cluster nodes.

## Conte Cluster Description

The Conte cluster consists of nodes with two 16 core Intel Xeon E5-2670 processors, two Xeon Phi accelerator cards and 64GB of memory. The 580 nodes are connected through an FDR Infiniband network at 40GB/s. Users are provided with a shared home directory as well as a share for widely-used applications; both are available from each node via NFS at approximately 2GB/s. These filesystems are used for all clusters at the university and accessed using IP over IB. The scratch filesystem used by the jobs is a Lustre 2.4 installation that can sustain up to 23GB/s and is connected via the above-mentioned IB network. Total capacity of the scratch filesystem is 1.4PB. Each node has a local filesystem and the OS installations are stateful and include a 2GB swap partition. All nodes run RedHat Enterprise Linux (RHEL) major version 6 and are regularly upgraded to the latest minor version; currently at 6.9. Along with default RHEL libraries, the environment also provides many other important pre-installed libraries and applications that can be loaded as needed. The nodes are administered using RedHat Kickstart installers integrated with Puppet configuration management software.

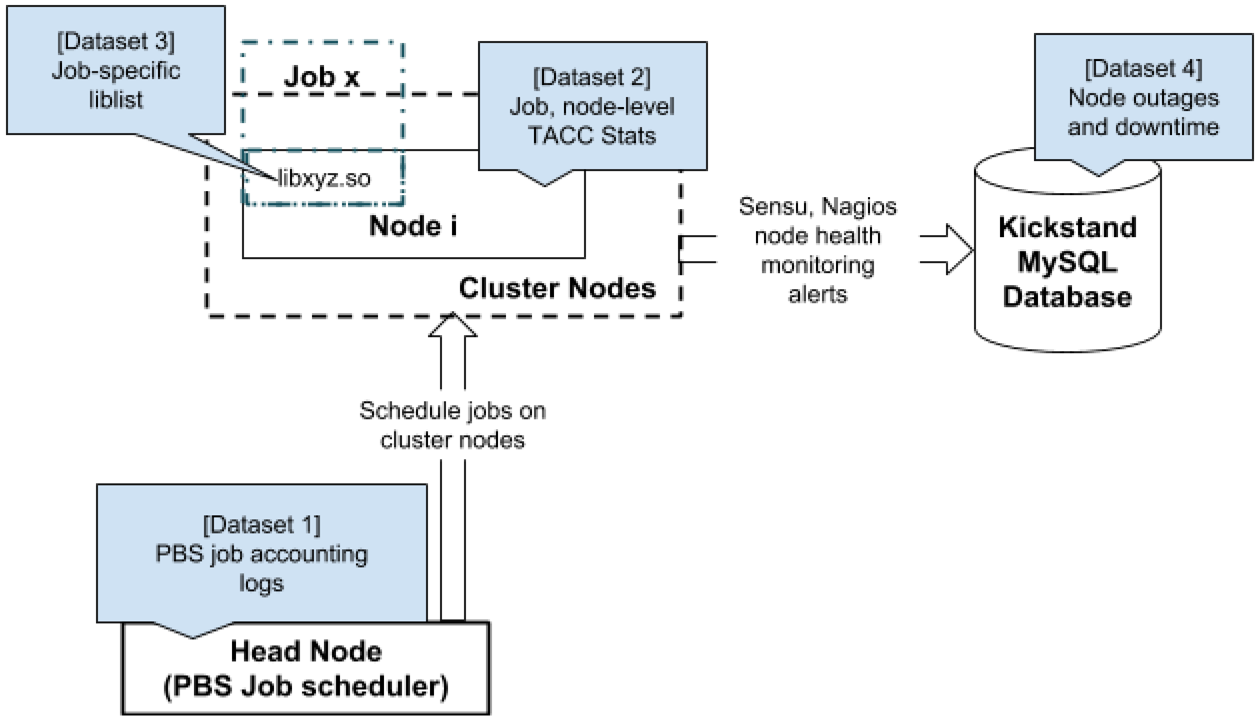
## Data Description

The FRESCO repository contains four categories of data collected over the period March 2015 through June 2017:

1. Accounting logs from the PBS job scheduler
2. TACC Stats performance data for each job
3. A listing of shared software libraries utilized by each job

4. Scheduled and unscheduled outages and downtime information for individual cluster nodes.

A broad overview of these four data sources is in the figure below, detailed explanations of each dataset follows.


[Dataset 3] Job-specific liblist
Job x
libxyz.so
Node i
[Dataset 2] Job, node-level TACC Stats
Cluster Nodes
Sensu, Nagios node health monitoring alerts
[Dataset 4] Node outages and downtime
Kickstand MySQL Database
Schedule jobs on cluster nodes
[Dataset 1] PBS job accounting logs
Head Node (PBS Job scheduler)

## Accounting Logs

Conte uses the open-source TORQUE implementation of the PBS job scheduling software. TORQUE maintains a record of a batch job as it moves through the job submission queue and eventual execution on the cluster. These records contain the event being recorded (e.g., queuing, job start, job end), corresponding timestamps, the submitting user or group, and resources requested and used. TORQUE accounting logs contain these records organized by date; each line in an accounting log file corresponds to a single event record for a job processed on that date. The accounting logs employ a fixed format for these records:

**<date> <timestamp>;<job event>;<jobID.node>;[<stats>]**

where **[<stats>]** is an array of <key>=<value> pairs where the respective keys can depend on the job event being recorded. For instance, a record of a queuing event would only contain queue=debug; while a job start event would include the qtime, start, and exec_host keys representing the time the job was queued, the time the job started running, and, the actual nodes the job was executed on, respectively. These records are anonymized to remove identifiable references to specific users, groups, queue or cluster nodes. Each record is then formatted into a comma separated list and stored in a CSV file organized by job execution month (***torque/yyyy-mm.csv***).

Job event types are encoded as single characters, according to the mapping table below:

| **Event Code** | **Event Type** | **Event Description** |
|---|---|---|
| **A** | abort | Job has been aborted by the server |
| **C** | checkpoint | Job has been checkpointed and held |
| **D** | delete | Job has been deleted |
| **E** | exit | Job has exited (either successfully or unsuccessfully) |
| **Q** | queue | Job has been submitted/queued |
| **R** | rerun | Attempt to rerun the job has been made |
| **S** | start | Attempt to start the job has been made (if the job fails to properly start, it may have multiple job start records) |
| **T** | restart | Attempt to restart the job (from checkpoint) has been made (if the job fails to properly start, it may have multiple job start records) |

A complete description of all the fields in these accounting log CSVs is in the table below:

| Field Name | Description |
|---|---|
| jobID | Anonymized version of identifier assigned by scheduler |
| timestamp | Timestamp of accounting record |
| jobevent | A one character job event code (refer to earlier table for interpretation of the codes) |
| user | User ID of the submitting user |
| group | Group of the submitting user |
| jobname | Descriptive job name specified by the user |
| queue | PBS queue the job was submitted to |
| account | Account to charge the job’s resources to |
| owner | Owner for this job (may be distinct from submitting user) |
| requestor | Account requesting this job (may be distinct from submitting user) |
| Exit_status | Exit status code (0 is success, others encode different error conditions) |
| ctime | Time job was created |
| qtime | Time job was queued |
| etime | Time job became eligible to run |
| start | Time job started execution |
| end | Time job ended execution |
| exec_host | Nodes and cores where job is scheduled to run |
| Resource_List.ncpus | Number of CPUs needed |
| Resource_List.neednodes | Number of nodes needed |
| Resource_List.neednodes.ppn | Number of processors per node requested |
| Resource_List.nodect | Limits on number of nodes to be used |
| Resource_List.nodes | Number of nodes requested for the job |

| | |
|---|---|
| Resource_List.nodes.ppn | Processors per node requested for the job |
| Resource_List.mem | Limits on the memory utilized |
| Resource_List.pmem | Limits on the peak memory utilization |
| Resource_List.walltime | Requested walltime for the job |
| Resource_List.naccesspolicy | Type of access policy |
| resources_used.cput | Amount of CPU time used by the job |
| resources_used.mem | Amount of memory used by the job |
| resources_used.vmem | Amount of virtual memory used by the job |
| resources_used.walltime | Walltime used by the job for completion |
| session | Session ID for the submitted job |

## TACC Stats Data

TACC Stats is a low-overhead infrastructure for collecting system-wide performance data for each node at regular intervals. This performance data is drawn from various sources such as, CPU usage, block device I/O, network usage, scratch and NFS filesystem I/O. TACC Stats data for a particular job is subsequently extracted from this raw data for each of the nodes the job ran on, based on the job accounting logs. The resulting cumulative data (for each device and execution node) is serialized into an ASCII file using Python's Pickle module. Our dataset contains aggregations of the unpickled versions[1] of each of these pickled job-level performance datasets decomposed into 16 individual CSV files (one for each device being monitored). Each CSV file contains performance data for the corresponding device type augmented with the job's unique ID, the identity of the node the data originated from as well as the timestamp of data collection. A short description of each of these device types being monitored can be found in the table below. Further details can be found in the paper *"Comprehensive Resource Use Monitoring for HPC Systems with TACC Stats", Evans, et al., Proceedings of the First International Workshop on HPC User Support Tools, 2014.*

| Device code | Device short description |
|---|---|
| block | Block device statistics (per device DEV) |

[1] "Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream is converted back into an object hierarchy.

| | |
|---|---|
| cpu | Scheduler accounting (per CPU) |
| ib_ext | Infiniband usage (per IB device) |
| intel_pmc3 | Intel Nehalem performance statistics |
| llite | Lustre FS (per mount) |
| lnet | Lustre network (per mount) |
| mdc | Metadata client (per mount) |
| mem | Memory usage (per node) |
| net | Network device usage (per device) |
| nfs | NFS file system usage (per device) |
| numa | NUMA statistics (per socket) |
| ps | Process statistics (per node) |
| sysv_shm | SysV shared memory segment usage (per node) |
| tmpfs | ram-backed file system usage (per node) |
| vfs | Dentry/file/inode/cache usage (per node) |
| vm | Virtual memory statistics (per node) |

Data for jobs that was executed during a month is aggregated into a single CSV file for each device type, for that month (***each tacc_stats/yyyy-mm subfolder contains upto 16 individual CSV files for that month***). The headers for these CSVs also contain individual performance parameter names for each of the monitored devices. An account of these data types where they are not immediately apparent follows:

**Device: vm**
pgpgin: unit=B
pgpgout: unit=B

**Device: llite**
read_bytes: unit=B
write_bytes: unit=B
direct_read: unit=B

direct_write: unit=B

**Device: block**
rd_sectors: unit=B
rd_ticks: unit=ms
wr_sectors: unit=B
wr_ticks: unit=ms
io_ticks: unit=ms
time_in_queue: unit=ms

**Device: ib_ext**
port_xmit_data: unit=B
port_rcv_data: unit=B

**Device: mem**
MemTotal: unit=B
MemFree: unit=B
MemUsed: unit=B
Active: unit=B
Inactive: unit=B
Dirty: unit=B
Writeback: unit=B
FilePages: unit=B
Mapped: unit=B
AnonPages: unit=B
PageTables: unit=B
NFS_Unstable: unit=B
Bounce: unit=B
Slab: unit=B

## Software Library Data

The library listing data contains a list of shared dynamic libraries that are currently loaded on a cluster node during a job's execution. This data is obtained by collating periodic snapshots of the shared libraries currently in use on a cluster node (identified by the *lsof* command). Our dataset contains aggregation of this data organized by month (***liblist/yyyy-mm.tsv***) in the tab-separated value (tsv) format. Each tsv file lists the anonymized username, job identifier, and, execution node, as well as, the library list and job exit status.

## Outage and Downtime Data

Information about Conte's infrastructure health is recorded in a MySQL database. This database contains events pointing to planned system outages, reboots or alerts from monitoring frameworks like Sensu [4] and Nagios [5]. We filter this data for our period of data collection (only the year 2015 in this case). The filtered data is available in a single csv file (***kickstand_2015.csv***) and contains the affected host, and, start and end timestamps for the outage event. Our current analysis utilizes these timestamps and host fields to determine downtimes of individual cluster nodes involved in a job's execution. A separate account of scheduled and unscheduled outages not recorded in the MySQL database, but reported in Purdue University's Research Computing group's news feeds can be found in the file ***Conte_outages.txt***.

## Accessing the FRESCO Data Repository

There are two approaches to accessing the FRESCO job scheduling and performance data.

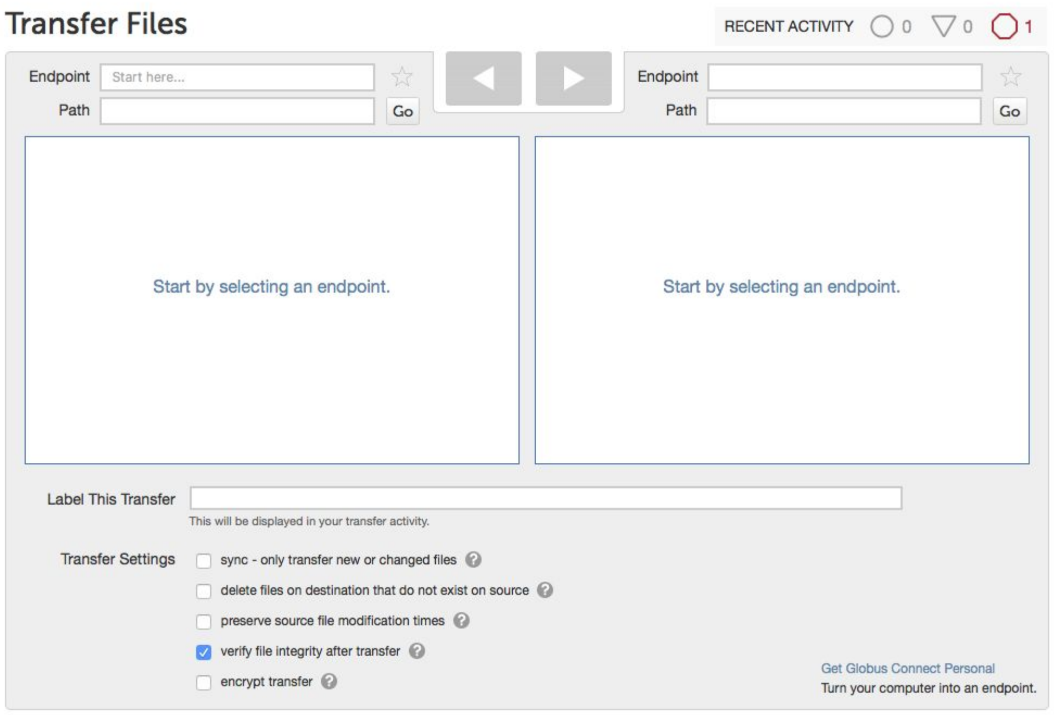
### Web Access

Interested users may open the FRESCO repository webpage (https://www.rcac.purdue.edu/fresco/) and download the data they are interested it using the links provided.

### Globus

We recommend using Globus data transfer to download the FRESCO data since the transfers need not be monitored once initiated, and will automatically resume even if your network connection is interrupted. Most universities and research labs should have a Globus endpoint that can be used to receive files. If you do not have access to a institutional Globus endpoint, you can setup your own Globus Connect Personal endpoint on your personal machine and transfer files to that endpoint. Instructions for setting up a Globus Connect Personal endpoint can be found here. Instructions for transferring files from the FRESCO endpoint to the desired target endpoint are below:

1. Access either your institution specific Globus transfer webpage, or the public Globus transfer webpage (https://www.globus.org/app/transfer). You should see a file transfer interface like the one below:



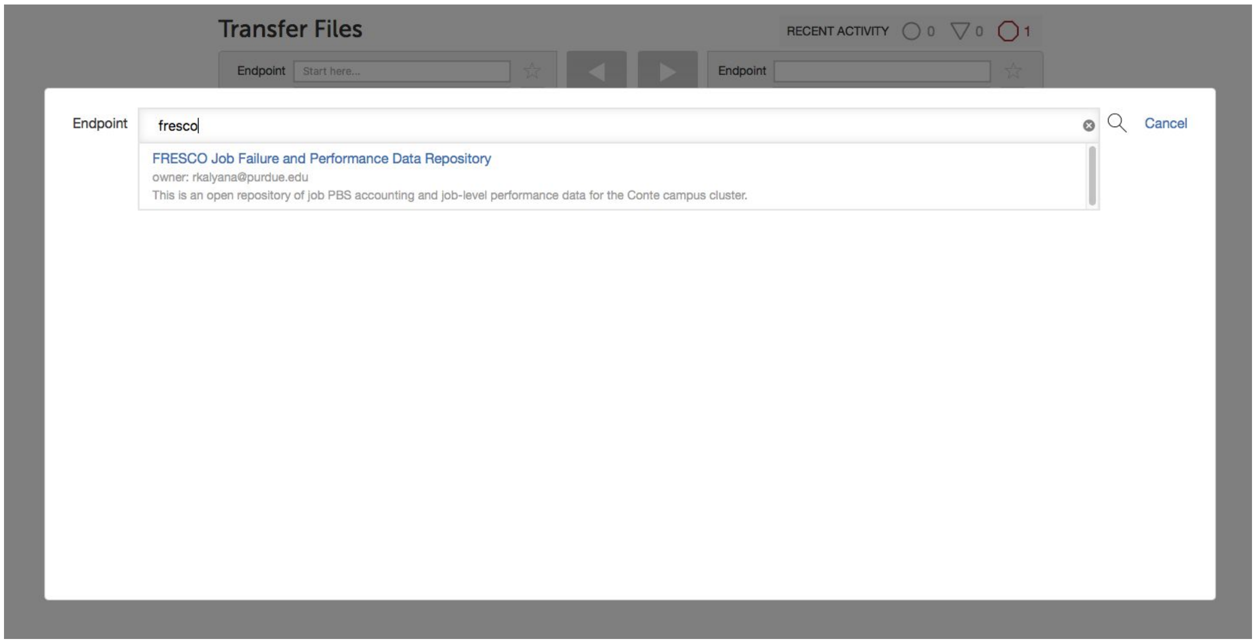
Label This Transfer

This will be displayed in your transfer activity.

Transfer Settings
- [ ] sync - only transfer new or changed files
- [ ] delete files on destination that do not exist on source
- [ ] preserve source file modification times
- [x] verify file integrity after transfer
- [ ] encrypt transfer

Get Globus Connect Personal
Turn your computer into an endpoint.

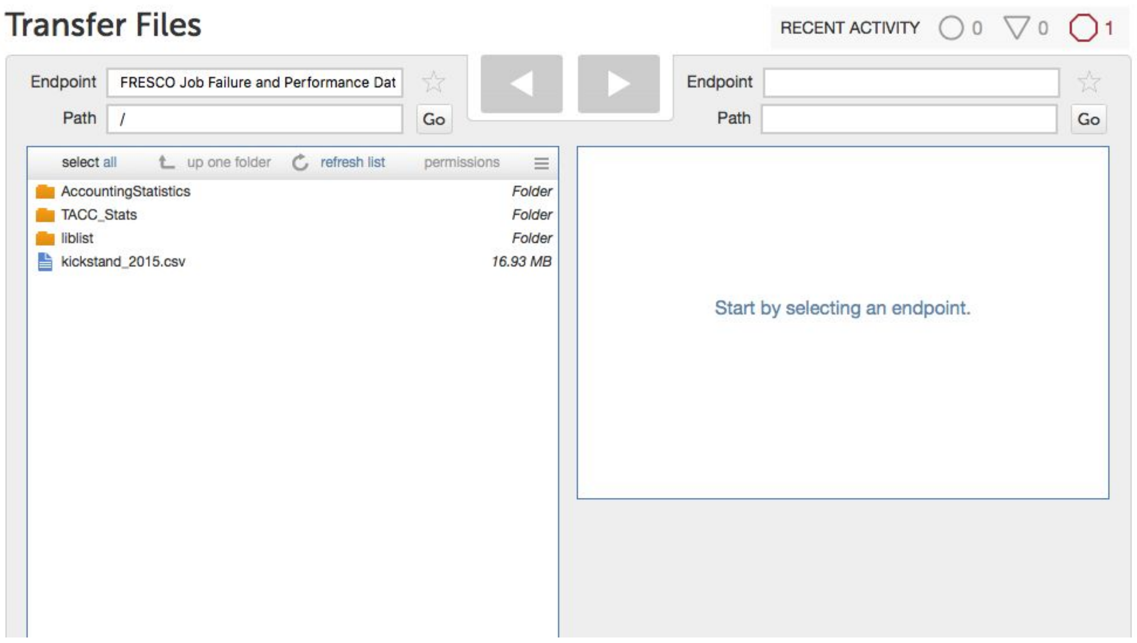
2. Click on the Endpoint: field (that says "Start here…") and search for the FRESCO public endpoint in the search interface that pops up as shown below:



Endpoint fresco

FRESCO Job Failure and Performance Data Repository
owner: rkalyana@purdue.edu
This is an open repository of job PBS accounting and job-level performance data for the Conte campus cluster.

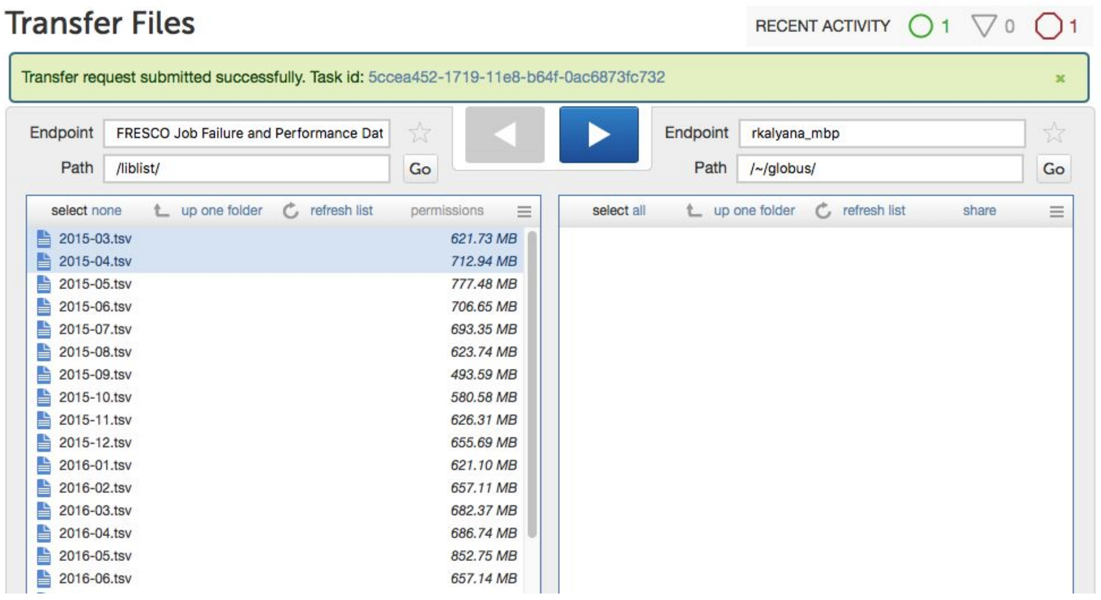
3. After selecting the Endpoint, you should be able to navigate through the repository contents.



Endpoint: FRESCO Job Failure and Performance Dat
Path: /

| | |
|---|---|
| AccountingStatistics | Folder |
| TACC_Stats | Folder |
| liblist | Folder |
| kickstand_2015.csv | 16.93 MB |

4. If you would like to transfer files to your endpoint, select that endpoint in the right-hand panel. You can select one or more files and click the arrow icon to start transferring files.



Transfer request submitted successfully. Task id: 5ccea452-1719-11e8-b64f-0ac6873fc732

Endpoint: FRESCO Job Failure and Performance Dat
Path: /liblist/

Endpoint: rkalyana_mbp
Path: /~/globus/

| | |
|---|---|
| 2015-03.tsv | 621.73 MB |
| 2015-04.tsv | 712.94 MB |
| 2015-05.tsv | 777.48 MB |
| 2015-06.tsv | 706.65 MB |
| 2015-07.tsv | 693.35 MB |
| 2015-08.tsv | 623.74 MB |
| 2015-09.tsv | 493.59 MB |
| 2015-10.tsv | 580.58 MB |
| 2015-11.tsv | 626.31 MB |
| 2015-12.tsv | 655.69 MB |
| 2016-01.tsv | 621.10 MB |
| 2016-02.tsv | 657.11 MB |
| 2016-03.tsv | 682.37 MB |
| 2016-04.tsv | 686.74 MB |
| 2016-05.tsv | 852.75 MB |
| 2016-06.tsv | 657.14 MB |

5. You can now close your browser window and check back later. You will also receive an email when the transfer has completed or if any issues were encountered.