Guide document for Bottleneck detection project:

**Analysis:**

This directory contains scripts and notebooks. The only important scrips in this directory are *remove\_cols* and *remove\_cols\_v3*. After collecting all the metrics from all the component, we might not need some of them and want to keep a subset of all collected metrics in the dataframe to use them further in the machine learning process.

These scripts read a csv file, we know the columns in the csv file, then removes some columns that we don’t need them, next same the new file.

In addition, we can have different severity levels for each anomaly, for example label 1,2,3 in the dataset can show same anomaly type but different severity. The next task that this script does is to merge these levels with each other ans saves another csv file for the merged dataset.

**Csv\_to\_protobuf\_bin:**

This directory contains script to post process collected csv file on the source and destination DTN to create a data frame for machine learning analysis.

For the machine learning training we need to run the transfer monitor application on both source and destination DTNs and config the application to collect and store the transfer data in a csv file with the desired label. After than we can run a transfer under the desired anomaly.

Once we run the transfer, the application discovers the transfer and collects data. The instance that which is run on the source collects data on the source side and the other instance collects data on the destination side. To be able to use the collected data for machine learning, we have to post process collected data.

First step is to merge these csv files collected from source and destination. The merge\_csv\_files.py script is doing this task. Next, we have to convert the csv file to a dataframe. csv\_to\_dataframe.py script does this part of the postprocessing. In the next step, we need to post process the created data frame using the scripts in the Analysis directory to remove redundant columns form the data frame.

**Dataset\_generator:**

*Parallel\_filesystem* directory contains the code of the monitoring application.

In *flask\_app\_agent* directory there are python scripts that are collecting lustre data from the OSSs. The flask app agent should be installed on OSSs. And there is a setup.sh that installs all the required packages fot the python and gunicorn as well. In the same setup.sh bash script there is an example command that shows how to use gunicon to run the flask application in the back ground. The monitoing agent sends a http request to this agent to collect ost metrics from the OSSs.

In the *BachScripts* directory there are bash scripts that they run the monitoring application on the destination and source DTNs and Inject anomaly in the system and finally run a transfer for specific time period. And stop all the monitoring agents, the injected anomaly and the running transfer after the specific time.

AgentMetricCollector contains the code for the monitoring agent.

The installation scripts are in the setupIns directory.

First we should run p3.8install.sh.

To run the code, we need to install python 3.8 and make it the default python interpreter

Then run install\_requirements.sh

This will install python requrements and java 11.

Next we have to run the chronyInstall.sh

This will install chrony and sync the time on all OSS servers with a chosen server. We should config the chrony by changing the chrony.conf file.

Install globus by running globusInstallation.sh.

Next if we want to use globus to run the transfer we have to install globus server and client on the DTN nodes. Config the globus port range in the globus\_port\_range.sh and copy this file and paste it in the /etc/profile.d directory on the DTN nodes. The example command how to run the globus server and using globus url copy is also in the globus\_port\_range.sh. There is a gridftp.conf in the globusConfig directory that should be used to config the globus server on DTN nodes.

Monitoring application:

The main script of the monitoring application is multi\_transfer\_parallel\_metric\_collector.py. It receives arguments as input parameter and runs other components as well as the java sender and receiver application. The configuration of the application is also in the Config.py script.

The multi\_transfer\_parallel\_metric\_collector.py first runs a transfer discovery component. The script of the transfer discovery component is in the discovery directory. The transfer\_discovery.py script is the main code for the transfer discovery component. It periodically monitors the active tcp connection and use the transfer\_validation\_strategy2.py strategy to filter out the non-transfer tcp connection and only monitors the tcp connections that are a transfer connection. The strategy for filtering out tcp connection is a white list strategy and the white list ip and ports that represent a transfer connection is in the Config file.

The multi\_transfer\_parallel\_metric\_collector.py also runs the TransferManager component. Whenever the transfer discovery module finds a valid tcp connection, it uses the TransferManager to create a monitoring agent process. For each valid transfer, there is a monitoring agent process that collect the performance metrics and store it in a file/send it to the cloud. Respectively whenever the transfer discovery finds an ended transfer, it users the TransferManager to remove the previously created monitoring agent process.

The TransferManager creates a new monitoring agent process by using stat\_collector\_thread.py script. This script is the code for collecting performance metrics from different components save it in a file/send it to the cloud, etc based on the configuration.

the stat\_collector\_thread scripts uses sub component called collector to collect metrics and convert them to different formats (string, josn, …). The collector scripts are in the collector directory. Each component of the system that we want to monitor has its own collector script.

There are also some metrics that can be shared among all the monitoring agent process such as memory or cpu usage of the DTN node. Also, there are some metrics such as ost metrics that are collected from the OSS servers that can be used by multiple monitoring agent processes. These metrics are stores in a cache and shared between processes as a shared memory. There are helper\_threads.py that collects such metrics and store them in the global\_vars.py script.

**Prototype:**

This directory contains some of the datasets collected from different clusters and python notebooks for running machine learning analysis.