Streaming Core-set Documentation

# General Description:

The server client module is a system which was design to cluster large amounts of data in streaming format.

The system is made of 3 main parts-

**The Server (server.py)**

**The Client (streamingClient.py)**

**The Worker (worker.py)**

The system works in the following manner:

Client

Server

The server receives the streamed information from the client,

It then distributes the information to the workers in 'CHUNK\_SIZE'.

Each worker creates a tree from the data it received.

The root of the tree is the core-set.

Upon request from the client, the server calls for the current core-set of each worker.

It then sends all the received core-sets to the summary worker- a dedicated worker which creates the final core-set, and returns the result to the client.  
  
  
The communication between the server/client and server/workers is done using codes defined in 'message\_codes.py' and use a socket communication.

# Main Components:

## Worker:

The worker handles the creation of the core-set.

Contains an instance of 'CoresetTreeAlgorithm' which creates a binary tree from the data and returns the core-set.

**Public Functions:**

**def** register\_and\_handle(self):

*Registers with the server and enters the send/receive loop.  
  
Connects to the server and tries to register. If successfully it  
starts waiting for commands, otherwise it stops and reports an  
error.*

## Server:

The server coordinates between all the registered workers and the client.

Listens to incoming commands from the client- when receiving data, divides it between the workers for processing.  
Upon request collects the unified core-set from all the workers and sends it back to the client.

**Public Functions:**

**def** main(self):

*initializes the server- makes the server prepared to register the workers and the client*.  
*after registration listens to incoming commands and performs accordingly*

**def** get\_summary\_from\_workers(self, client\_socket):

*Queries each worker for the current summary  
  
Each registered worker is asked for the current summary  
via codes.GETUNIFIED. Then the function waits till all the  
chunks arrive in a for loop. When the data is ready  
it is serialized and sent to the client*

## streamingClient:

Streaming client mimics the client.

It generates the data and streams it to the server, then requests the summary.

**Public functions:**

**def** run\_client(self):

*starts the client. creates a database and sends it to the server, then asks for the summary (coreset)*

**def** get\_summary\_points(self):

*requests and handles the summary (coreset) from the server*

## CoresetTreeAlgorithm:

Exists within each worker-

Works by creating a binary tree, where each node is of size 'CORESET\_SIZE'.

The leaves are the original data, and each level above provides a smaller and better representation of the data.   
The root holds the best representation of all the leaves: the core-set.

Has a configurable merging algorithm – 'CORESET-ALGORITHM'.

**Public functions:**

**def** add\_points(self, points):

*Adds a set of points to the stream.  
  
If the set is larger than coreset\_size, it is split  
into several sets and a coreset is constructed on each set.***:param** *points: input data***:return***: none*

**def** get\_unified\_coreset(self):

*merges the tree and returns the unified coreset (root)*

# Utility Components:

## array\_util:

Defines some helper functions for arrays- e.g. split to chunk size

## client\_runner:

Creates and runs a client.

## connection:

Defines the socket connection- handles the Receive and Send functions while dividing the data into smaller 'PACKET\_SIZE' packets in order to avoid the socket max data limit.

## connection\_data:

Defines the connection information: Server IP, Worker port and Client port.

## connection\_util:

Defines some helper functions which wrap the serialization and sending of the information.

## connectionListener:

Listens to a current connection

## CreateDatabase:

Creates a csv file of a sparse matrix to be used as a database for the program.

## message:

Defines the message class used in socket communication

## message\_codes:

Defines the codes used in the communication between the client/server and server/worker.

## server\_runner:

Creates and runs a server.

## Stack:

Implementation of a stack data structure.

## summary\_worker\_runner:

Creates and runs a summary worker.

## worker\_runner:

Creates and runs a worker.

# Testing Components:

## WorkManager:

Simulates the entire working system.

Sets a server, registers a number of workers (defined in the WorkManager constructor) and a summary worker, and connects a clients.

The client creates a DB and sends the data to the server.  
Then it asks for the summary which the server provides, and exits.

## simpleCoreset:

Provides a very simplistic method for merging nodes in the tree.

Contains a simple coreset algorithm - return a random portion of the data in size of coreSetSize