Clustering Algorithm Implementation and Explanation

To read more about Clustering please check our paper at https://www.usenix.org/conference/fast22/presentation/kisous

"The what, The from, and The to: The Migration Games in Deduplicated Systems".

In 20th USENIX Conference on File and Storage Technologies (FAST 22), 2022.

Authors:

Roei Kisous and Ariel Kolikant, ***Technion - Israel Institute of Technology;***

Abhinav Duggal, ***DELL EMC;***

Sarai Sheinvald, ***ORT Braude College of Engineering;***

Gala Yadgar, ***Technion - Israel Institute of Technology***

Guide Author: Roei Kisous

Contributor: Shalev Kuba

1) Introduction:

The purpose of this guide is to explain how to recreate the experiments done with the HC algorithm.

The algorithm creates a bit appearance matrix based on the system snapshots provided, where each file represents a row and each column represents a block, 1 is set if the file contains the block, 0 otherwise.

With this matrix we generate our dissimilarity matrix, where each row and column correspond to a file, and the values represent the Jaccard distance between them. As the matrix is symmetric, only the lower triangular is used for the algorithm; the upper triangular is used to restore the original values if multiple runs are performed.

Using the parameters set, these value are changed in accordance with the explanation in the article. The algorithm performs hierarchical clustering and outputs the results into results files.

2) Files you will need:

1) CommandLineParser.cpp/hpp
2) AlgorithmDSManager.cpp/hpp
3) HierarchicalClustering.cpp/hpp
4) Node.cpp/hpp
5) Utility.cpp/hpp
6) Main.cpp
7) Makefile

1. Run the Makefile to build the Clustering algorithm code.
2. Create volumes that are compatible with the used standard. See the volume standard section.
3. Run HC using examples. This is explained in the running HC section.
4. Run calc using the migration plan as input. See the section on running Calc.

3) Volume Standard:

The volumes' files representing the initial state of a volume within the system. It tells us what files are in the volume and what blocks those files contain. Blocks have unique SNs (serial numbers) shared across volumes.

1) Header:

```
#Output type: block-level
#Input files: 0002
#Target depth: 10
#Num files: 36
#Num directories: 29
#Num Blocks: 268
```

2) File lines:

$$\underbrace{F}_{\text{Object type}},\underbrace{0}_{\text{SN}},\underbrace{002\_281474976710693}_{\text{File ID}},\underbrace{0}_{\text{Parent Directory SN}},\underbrace{11}_{\text{Num of base objects}},\underbrace{0}_{\text{Base Object SN}},\underbrace{16046}_{\text{Base Object size}},\underbrace{1}_{\text{Base Object SN}},\underbrace{4096}_{\text{Base Object size}},\ldots$$

3) Block lines:

$$\underbrace{B/P}_{Object\ type},\underbrace{77}_{SN},\underbrace{9029760f86}_{Object\ ID},\underbrace{5}_{Shared\ by\ num\ files},\underbrace{26}_{file\ SN},\underbrace{31}_{file\ SN},\underbrace{57}_{file\ SN},\ldots$$

4) Folders:

$$\underbrace{D/R}_{Object\ type},\underbrace{2}_{SN},\underbrace{002\_281474976710694}_{Directory\ ID},\underbrace{0}_{Parent\ Directory\ SN},\underbrace{2}_{Num\ of\ Subdirs},\underbrace{4}_{Num\ of\ Files},\underbrace{3}_{Dir\ SN},\underbrace{5}_{Dir\ SN},\underbrace{17}_{File\ SN},\underbrace{21}_{File\ Sn},\ldots$$

An example file might look like this:

# Output type: block-level								
# Input files: 0001	0002	0003	0004	0005				
# Target depth: 1								
# Num files:								
# Num directories: 0								
# Num Blocks: 3								
F	0	ID1	0	2	0	368	1	10555
F	1	ID2	1	2	0	368	1	10555
F	2	ID3	2	2	0	368	2	97973
F	3	ID4	3	2	0	368	2	97973
F	4	ID5	4	2	0	368	1	10555
B	0	zzzzzzzzzzz	5	0	1	2	3	4
B	1	00000219000	3	0	1	4		
B	2	00000f1f4f	2	2	3			

Where, for example, file 2 contains 2 blocks (SNs) – 0 of size 368 and 2 of size 97973.

Note that across different volumes of the same execution the following must be persistent:

1. SN of a block
2. Fingerprint of a block

The file's ID is a string that describes the file system snapshot and can be ignored. We refer to each file by its unique SN.

It is possible that some blocks' SNs might be missing because files on this volume do not contain the block, as we are splitting a volume with sequential block SNs into a few.

4) Running HC:

Command line arguments

1) -workloads: full path of the volumes to load, list of strings.
2) -fps: number of min hash fingerprints (input 'all' for all fps) - int or 'all'
3) -WT: W_T values, list of doubles.
4) -lb: use load balancing (optional, default false)
5) -seed: seeds for the algorithm, list of int
6) -gap: pick random value from values in this gap - list of double
7) -lb_sizes: a list of clusters' requested sizes - list of int, must sum to 100 (optional, default is even distribution)
8) -eps: % to add to system's initial size at every iteration - int (mandatory if -lb is used, default is 5)
9) -clusters: number of clusters to output (optional, default is same number of input volumes)
10) -output_path_prefix: relative path + prefix of the result files

Output

The experiments don't include just one, but multiple runs. For each of these runs, their parameters and their results, a result file is generated, which can be sent directly to the calculator. When running this result file with the calculator, the cost will be calculated for the filtered system. In order to calculate the original cost, one must change the paths in the result file to the original volume locations (the first column in the output file). The 'invisible' header of the output files is:

Path to volume	SN of the initial volume's files	SN of the final volume's files	Whether to use for system's size calculation – always 1

The results will be generated in the -output_path_prefix specified location with the following suffix:

WTXX_SXX_GXX_EXX_XX.csv

1. WTXX – T and then the W_T for this plan
2. SXX – S and then the seed
3. GXX – G and then the gap
4. EXX – E and then the eps
5. XX – lb or no lb

Make sure the folder exists before.

Examples

- A rather simple example is to use the following command:

./hc -workloads example/vol1.csv example/vol2.csv -fps all -WT 1 -lb -seed 0 -gap 1 -output_path_prefix results/example1

It will result (results/example1_WT1.00_S0.00_G1.00_E5_lb.csv.csv) with a single result csv corresponding to the two volumes specified, while using all of the blocks available, $W_T = 1$, seed 0, 50% load balancing and gap 1%:

/nfs_share/storage-simulations/Roei/Gala/HC/example/vol1.csv	0-1-2-3-4-5-6-7-8-9	0-1-2-4-5-6-7-8-10-11-12-13-14-15-16-17-18-19	1
/nfs_share/storage-simulations/Roei/Gala/HC/example/vol2.csv	10-11-12-13-14-15-16-17-18-19	3-9	1

- Another example might be for multiple runs using:

./hc -workloads example/vol1.csv example/vol2.csv -fps all -WT 0 1 -lb -seed 0 10 -gap 0 1 -eps 5 -output_path_prefix results/example2

It performs the clustering process on every combination of parameters given, 8 in our case.

It leverages the data structures (dissimilarity matrix, appearances matrix) already created for the first execution in order to run the others, saving time generating those data structures at every executions.

This results are the files with the prefix example2 in results folder.

- A generalized load balancing example might be:

./hc -workloads example/vol1.csv example/vol2.csv -fps all -WT 0 1 -lb -seed 0 10 -gap 0 1 -lb_sizes 30 70 -output_path_prefix results/example3

It performs the clustering process on every combination of parameters given as before.

As opposed to before, the load balancing constrain is set to 30% and 70%, which means that the two final volumes are needed have those capacities of the system size.

This results are the files with the prefix example3 in results folder.

As you can see there, the migration plan is significantly different the previous run.

- In case you want to add a volume, you might use:

./hc -workloads example/vol1.csv example/vol2.csv example/empty.csv -fps all -WT 0 1 -lb -seed 0 10 -gap 0 1 -output_path_prefix results/example4

Adding an empty volume in the workloads tag forces the algorithm to produce 3 clusters, as in one of the output files:

/nfs_share/storage-simulations/Roei/Gala/HC/example/empty.csv		3-9	1
/nfs_share/storage-simulations/Roei/Gala/HC/example/vol1.csv	0-1-2-3-4-5-6-7-8-9	1	1
/nfs_share/storage-simulations/Roei/Gala/HC/example/vol2.csv	10-11-12-13-14-15-	0-2-4-5-6-7-8-10-11-12-13-14-15-16-17-18-19	1

	16-17-18-19		

Any example we showed, can be executed without -lb to receive an output without load balancing.

Paper evaluation

Obtaining the paper's results requires that you run the instance with:

1) -WT: 0, 0.2, 0.4, 0.6, 0.8, 1
2) -seed: 0, 37, 41, 58, 99, 111, 185, 199, 523, 666
3) -gap: 0.5, 1, 3

Given T_{max} and μ, choose the best possible result.

- Legal results for the relaxed version are chosen from all those that satisfy $actual\ sampled\ traffic \leq T_{max}$.
- Legal results for the regular version are chosen from all those that satisfy $actual\ sampled\ traffic \leq T_{max}\ and\ actual\ sampled\ balance \geq \frac{\frac{100}{|V|}-\mu}{\frac{100}{|V|}+\mu}$.

For either version, the best result for T_{max} and μ, is the one where the system's deletion is the greatest among the legal results.

5) Running Calc:

When we create a migration plan, we want to be able to simulate it on the system and determine how much data has been deleted and moved in practice.

run calc as such:

./calc -file [the migration plan in calc format] -output [where you want to save the output]

The migration plan in calc format is explained in the output section. The calculator would create, at the folder given under the output flag, a csv file looking like this:

workload	initial size B	initial size %	traffic B	deletion B	final size B	final size %	total traffic B	total traffic %	total deletion B	total deletion %	lb score
Vol1_sampled.csv	147699	87.8805	0	13859	133840	82.3484					
Vol2_sampled.csv	20369	12.1195	8320	0	28689	17.6516					
							8320	4.95038	5539	3.29569	0.214353

once you change the sampled volumes path to the original ones (in the migration plan's file) and run, you will receive the equivalent calculations for the original volumes under the migration plan.

The header components are:

- Workload: the volume file used for the simulation.
- Initial size B: the initial size of the volume in bytes
- Initial size %: the initial percentage of the volume’s size from the total system's initial size.
- Traffic B: the size in bytes moved into the volume under this migration plan simulation.
- Deletion B: the size in bytes deleted from the volume under this migration plan simulation.
- final size B: the final size of the volume in bytes.
- final size %: the final percentage of the volume’s size from the total system's final size.
- total traffic B: the total amount of bytes that have been moved under the simulation.
- total traffic %: what percentage of the **initial system size** has been moved.
- total deletion B: the total amount of bytes that have been deleted under the simulation.
- total deletion %: what percentage of the **initial system size** has been deleted.
- lb score: the load balancing score of the system after the migration.