Cost Calculator Implementation and Explanation

To read more about Calculator 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

Introduction

For evaluating the nature of a specific migration plan, we implemented a calculator that outputs the costs and requirements.

From this plan, the calculator provides the traffic consumption, score achieved, and deletion obtained while also providing the initial and final size of each volume with specific traffic and deletion. Each volume is iterated over and its incoming traffic and resulting deletion is calculated. The final traffic is equal to the sum of all traffic calculated, and the final deletion is equal to the sum of all deletion minus the sum of all traffic.

We added a results cache to save time, so that in case a migration plan needs to be calculated again (which we might not know about), it is not calculated again and instead the results are retrieved from the cache. The migration plan's hash is inserted as a key into the cache database with the corresponding results. The hash is recalculated and searched in the database when trying to find this plan again. It is returned if it is found, otherwise it is recalculated. The results will be fetched if the results are in the db specified in the -cache_path tag and the -no_cache tag was not used.

Our file-based lock allows the calculator to run on multiple threads concurrently without user interaction. There is no restriction on the number of threads the user may run as long as he does not delete the txt file created under the folder in which the database is stored, this is the file lock.

Furthermore, to test our plan, we use the calculator both on the sampled system (for real-time evaluation) and on the original system (for static evaluation), as shown below.

Files

1. Lock.cpp/h – Implementation of a file lock to simulate a global lock for multiple concurrent users.
2. CommandLineParser.cpp/h - implementation of a command line parser similar to Python.
3. main.cpp – this is the calculator itself.
4. Makefile – a makefile to compile the project.

Command line arguments and packages

For this project to compile and run, you must install sqlite3 and pthread and run 'make', then run the 'calc' exe with the following command line arguments:

1. -file – the input file described below.
2. -output - optional, the name of the output file, described below.
3. -cache_path - the full path to a sqlite3 database (.db).
4. -no_cache - optional, do not use the cache (useful for small volumes).
5. -lb_sizes - optional, a list of the requested sizes for each final volume. When this argument is omitted, it is set to perfect load balancing.

Note that if more than 20 volumes are used, the cache will not be used regardless of the -no_cache option.

-file argument

Migration plan generated by an algorithm (HC, ILP, Greedy, etc.) in the following format:

Path to the volume	Original files' SN in the volume separated by '-'	Final files' SN in the volume separated by '-'	Whether to use for size calculation (0/1, we always used 1)

For instance, a migration plan with two volumes might look like this (without headers):

vol1.csv	0-1-2-3-4	0-2-4-6-8	1
vol2.csv	5-6-7-8-9	1-3-5-7-9	1

- Files' SN order is irrelevant.

According to the table, files 0-4 were in vol_1 and files 5-9 were in vol_2, but now evens are in vol_1 and odds are in vol_2.

Volumes have the following format, with blocks' SN shared between them:

1) Header:

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

2) File lines:

F (Object type), 0 (SN), 002_281474976710693 (File ID), 0 (Parent Directory SN), 11 (Num of base objects), 0 (Base Object SN), 16046 (Base Object size), 1 (Base Object SN), 4096 (Base Object size), ...

3) Block lines:

B/P (Object type), 77 (SN), 9029760f86 (Object ID), 5 (Shared by num files), 26 (file SN), 31 (file SN), 57 (file SN), ...

For example:

# Output type: block-level								
# Input files: 0001	0002	0003	0004	0005				
# Target depth: 1								
# Num files: 5								
# 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			

In file 2 there are two blocks - 0 of 368 bytes, and 2 of 97973 bytes.

Note that across different volumes 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.

Output

The output is given in the following format:

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
Vol_1.csv	num	num	num	num	num	num					

Vol_2.csv	num	num	num	num	num	num					
							num	num	num	num	num

- 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.

In the absence of -no_cache, the result is inserted into the cache.

Examples

The first use case will be checking its costs on a sampled system and the second will be on the original system. The only difference will be in the volumes' paths. Both will require perfect load balancing - 50% at each volume. Use '-lb_sizes 70 30', for example, to simulate a 70% 30% capacity distribution between the volumes.

The first two examples were created from the UBC50 trace. The trace was split into 5 traces of 10 consecutive files each. The original system in our example is k=18, and the sampled one is k=20 (so we can easily track the results).

Running a migration plan on a sampled system

With a migration plan generated by either of the algorithms given as 'example_migration_sample.csv':

example/example_sampled_vol1.csv	0-1-2-3-4-5-6-7-8-9	0-3-5-6-7-8-9	1
example/example_sampled_vol2.csv	10-11-12-13-14-15-16-17-18-19	10-11-12-13-14-15-16-17-18-19-1-2-4	1

Using the following command, we can now check the cost of our migration plan on the sampled system:

./calc -file example/example_migration_sample.csv -output example/output_sampled.csv -cache_path example/example.db -no_cache

Following the explanation above, here is the output_sampled.csv:

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
example_sampled_vol1.csv	147699	87.8805	0	13859	133840	82.3484					
example_sampled_vol2.csv	20369	12.1195	8320	0	28689	17.6516					
							8320	4.95038	5539	3.29569	0.214353

Running a migration plan on an original system

The only difference in the sampled system is the paths of volumes. The input file is as follows:

example/example_original_vol1.csv	0-1-2-3-4-5-6-7-8-9	0-3-5-6-7-8-9	1
example/example_original_vol2.csv	10-11-12-13-14-15-16-17-18-19	10-11-12-13-14-15-16-17-18-19-1-2-4	1

And by running:

./calc -file example/example_migration_original.csv -output example/output_original.csv -cache_path example/example.db

We receive:

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
example_original_vol1.csv	374589	39.862	0	0	374589	39.862					
example_original_vol2.csv	565125	60.138	0	0	565125	60.138					
							0	0	0	0	0.662843

Which is different than our sampled result.