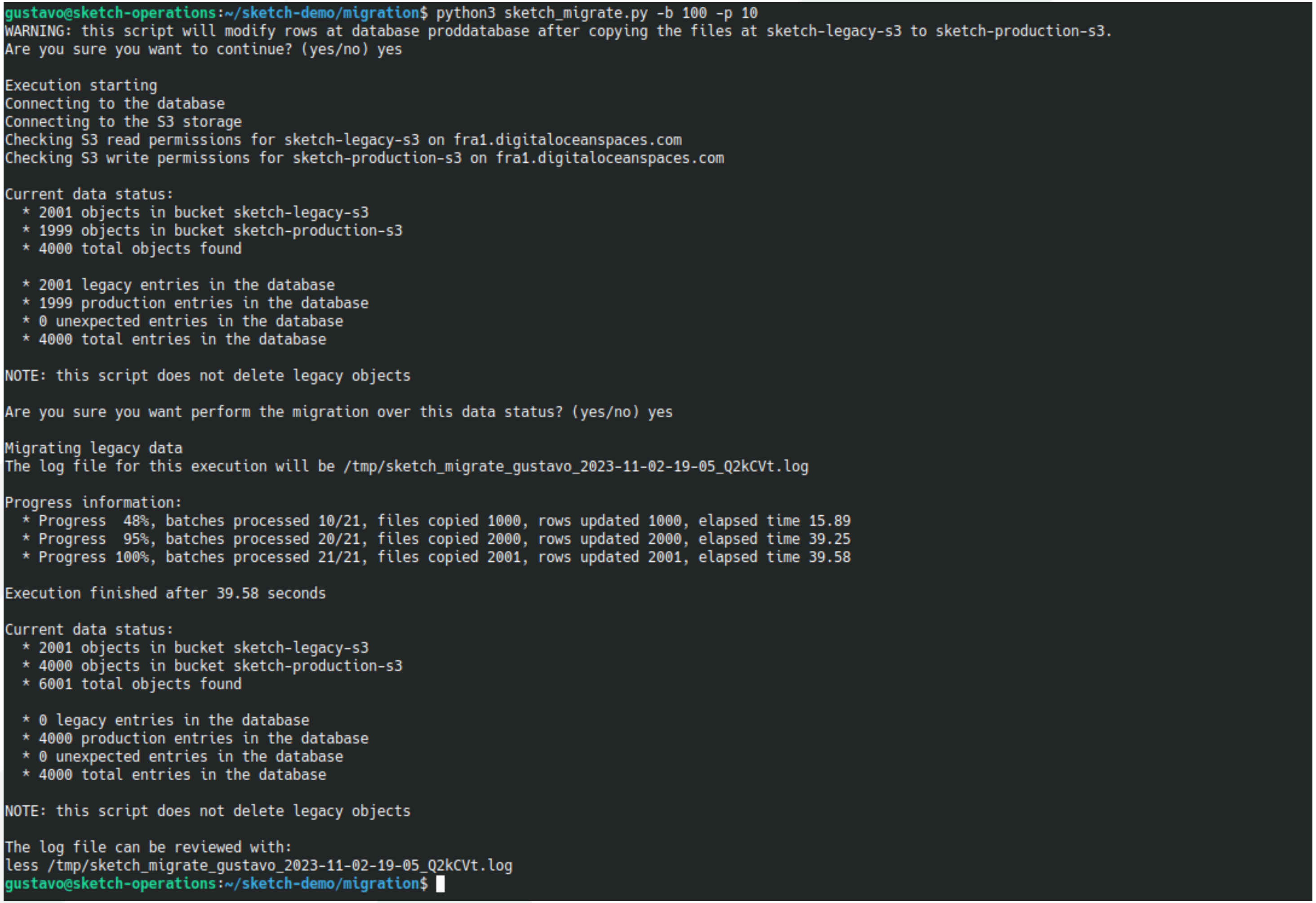
Sketch demo

Project highlights – rev 01

Gustavo Homem

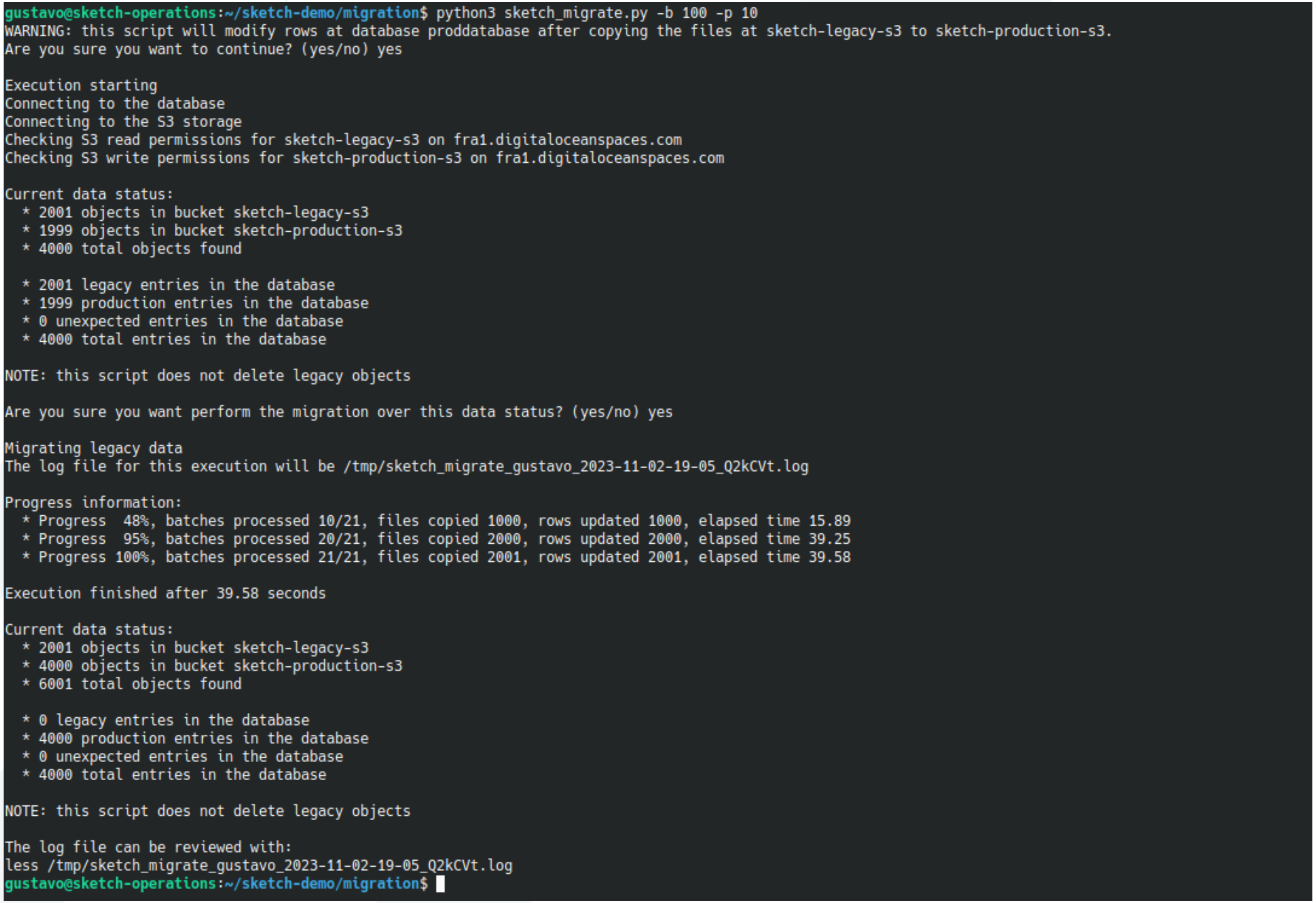
homem.gustavo@gmail.com
October 2023

TLDR - migration tool



```
gustavo@sketch-operations:~/sketch-demo/migration$ python3 sketch_migrate.py -b 100 -p 10
WARNING: this script will modify rows at database proddatabase after copying the files at sketch-legacy-s3 to sketch-production-s3.
Are you sure you want to continue? (yes/no) yes

Execution starting
Connecting to the database
Connecting to the S3 storage
Checking S3 read permissions for sketch-legacy-s3 on fra1.digitaloceanspaces.com
Checking S3 write permissions for sketch-production-s3 on fra1.digitaloceanspaces.com

Current data status:
  * 2001 objects in bucket sketch-legacy-s3
  * 1999 objects in bucket sketch-production-s3
  * 4000 total objects found

  * 2001 legacy entries in the database
  * 1999 production entries in the database
  * 0 unexpected entries in the database
  * 4000 total entries in the database

NOTE: this script does not delete legacy objects

Are you sure you want perform the migration over this data status? (yes/no) yes

Migrating legacy data
The log file for this execution will be /tmp/sketch_migrate_gustavo_2023-11-02-19-05_Q2kCVt.log

Progress information:
  * Progress  48%, batches processed 10/21, files copied 1000, rows updated 1000, elapsed time 15.89
  * Progress  95%, batches processed 20/21, files copied 2000, rows updated 2000, elapsed time 39.25
  * Progress 100%, batches processed 21/21, files copied 2001, rows updated 2001, elapsed time 39.58

Execution finished after 39.58 seconds

Current data status:
  * 2001 objects in bucket sketch-legacy-s3
  * 4000 objects in bucket sketch-production-s3
  * 6001 total objects found

  * 0 legacy entries in the database
  * 4000 production entries in the database
  * 0 unexpected entries in the database
  * 4000 total entries in the database

NOTE: this script does not delete legacy objects

The log file can be reviewed with:
less /tmp/sketch_migrate_gustavo_2023-11-02-19-05_Q2kCVt.log
gustavo@sketch-operations:~/sketch-demo/migration$ 
```

TLDR - migration tool



```
gustavo@sketch-operations:~/sketch-demo/migration$ python3 sketch_migrate.py -b 100 -p 10
WARNING: this script will modify rows at database proddatabase after copying the files at sketch-legacy-s3 to sketch-production-s3.
Are you sure you want to continue? (yes/no) yes

Execution starting
Connecting to the database
Connecting to the S3 storage
Checking S3 read permissions for sketch-legacy-s3 on fra1.digitaloceanspaces.com
Checking S3 write permissions for sketch-production-s3 on fra1.digitaloceanspaces.com

Current data status:
  * 2001 objects in bucket sketch-legacy-s3
  * 1999 objects in bucket sketch-production-s3
  * 4000 total objects found

  * 2001 legacy entries in the database
  * 1999 production entries in the database
  * 0 unexpected entries in the database
  * 4000 total entries in the database

NOTE: this script does not delete legacy objects

Are you sure you want perform the migration over this data status? (yes/no) yes

Migrating legacy data
The log file for this execution will be /tmp/sketch_migrate_gustavo_2023-11-02-19-05_Q2kCVt.log

Progress information:
  * Progress  48%, batches processed 10/21, files copied 1000, rows updated 1000, elapsed time 15.89
  * Progress  95%, batches processed 20/21, files copied 2000, rows updated 2000, elapsed time 39.25
  * Progress 100%, batches processed 21/21, files copied 2001, rows updated 2001, elapsed time 39.58

Execution finished after 39.58 seconds

Current data status:
  * 2001 objects in bucket sketch-legacy-s3
  * 4000 objects in bucket sketch-production-s3
  * 6001 total objects found

  * 0 legacy entries in the database
  * 4000 production entries in the database
  * 0 unexpected entries in the database
  * 4000 total entries in the database

NOTE: this script does not delete legacy objects

The log file can be reviewed with:
less /tmp/sketch_migrate_gustavo_2023-11-02-19-05_Q2kCVt.log
gustavo@sketch-operations:~/sketch-demo/migration$ 
```

Confirm you want to write into prod

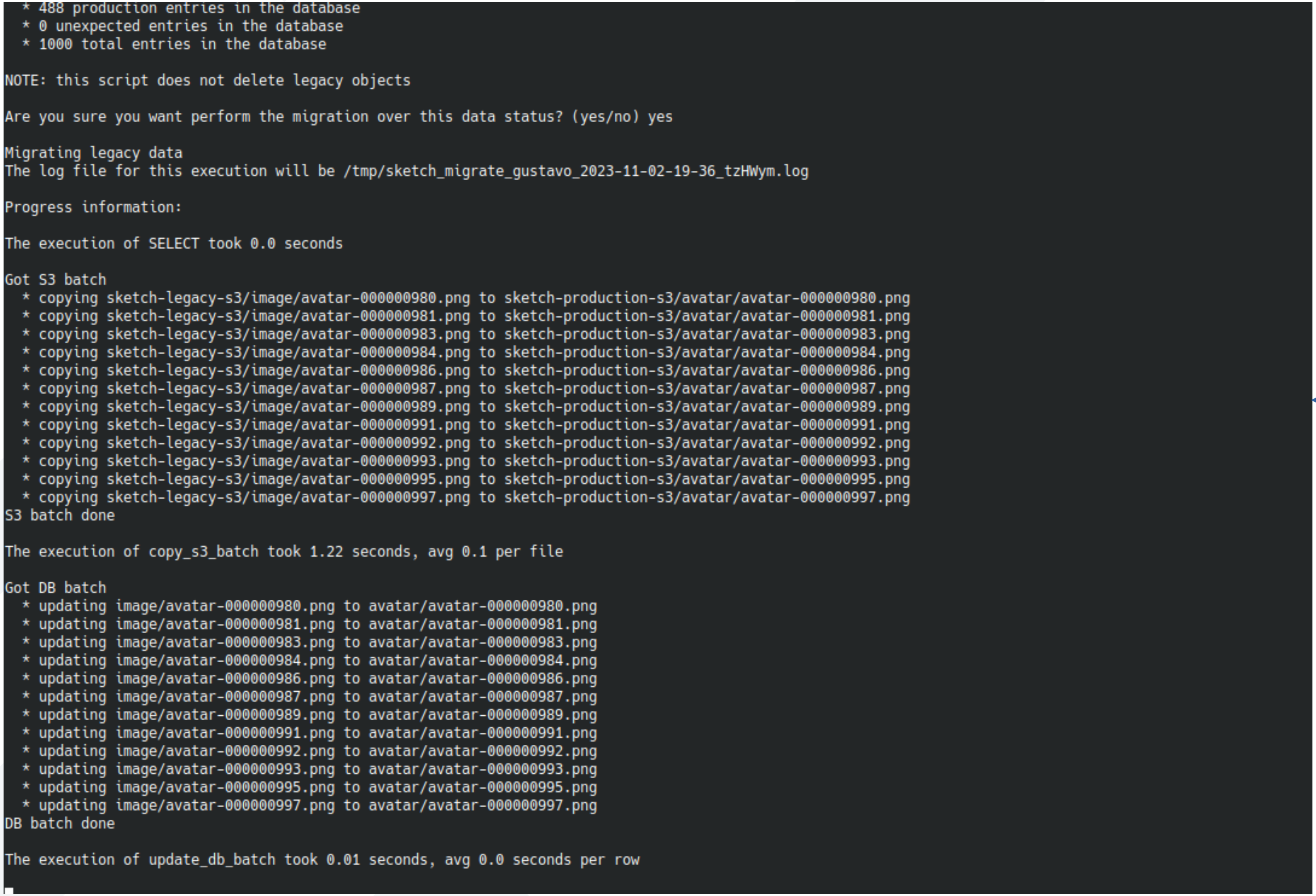
Confirm the initial situation makes sense

Check progress

Review the final situation

Be aware of the log file

TLDR - migration tool in verbose mode

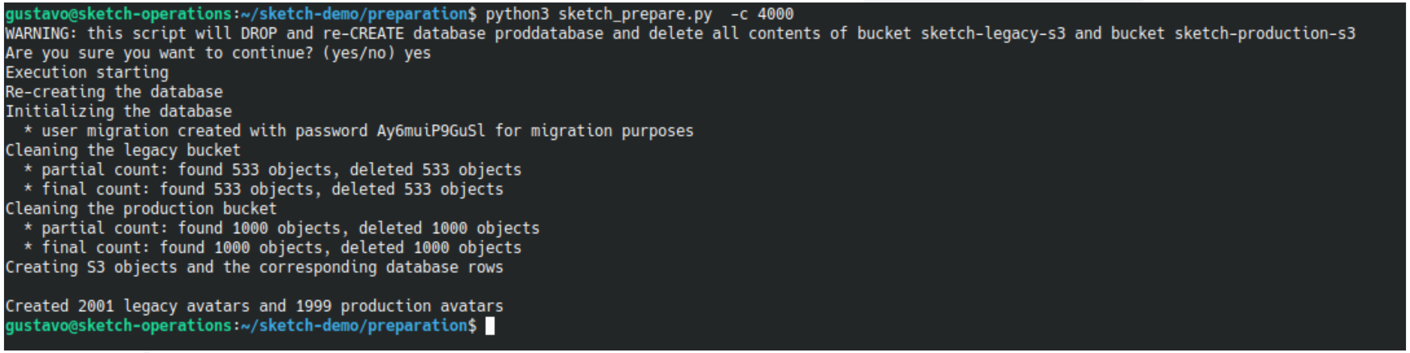


```
  * 488 production entries in the database
  * 0 unexpected entries in the database
  * 1000 total entries in the database

NOTE: this script does not delete legacy objects

Are you sure you want perform the migration over this data status? (yes/no) yes

Migrating legacy data
The log file for this execution will be /tmp/sketch_migrate_gustavo_2023-11-02-19-36_tzHWym.log

Progress information:

The execution of SELECT took 0.0 seconds

Got S3 batch
  * copying sketch-legacy-s3/image/avatar-000000980.png to sketch-production-s3/avatar/avatar-000000980.png
  * copying sketch-legacy-s3/image/avatar-000000981.png to sketch-production-s3/avatar/avatar-000000981.png
  * copying sketch-legacy-s3/image/avatar-000000983.png to sketch-production-s3/avatar/avatar-000000983.png
  * copying sketch-legacy-s3/image/avatar-000000984.png to sketch-production-s3/avatar/avatar-000000984.png
  * copying sketch-legacy-s3/image/avatar-000000986.png to sketch-production-s3/avatar/avatar-000000986.png
  * copying sketch-legacy-s3/image/avatar-000000987.png to sketch-production-s3/avatar/avatar-000000987.png
  * copying sketch-legacy-s3/image/avatar-000000989.png to sketch-production-s3/avatar/avatar-000000989.png
  * copying sketch-legacy-s3/image/avatar-000000991.png to sketch-production-s3/avatar/avatar-000000991.png
  * copying sketch-legacy-s3/image/avatar-000000992.png to sketch-production-s3/avatar/avatar-000000992.png
  * copying sketch-legacy-s3/image/avatar-000000993.png to sketch-production-s3/avatar/avatar-000000993.png
  * copying sketch-legacy-s3/image/avatar-000000995.png to sketch-production-s3/avatar/avatar-000000995.png
  * copying sketch-legacy-s3/image/avatar-000000997.png to sketch-production-s3/avatar/avatar-000000997.png
S3 batch done

The execution of copy_s3_batch took 1.22 seconds, avg 0.1 per file

Got DB batch
  * updating image/avatar-000000980.png to avatar/avatar-000000980.png
  * updating image/avatar-000000981.png to avatar/avatar-000000981.png
  * updating image/avatar-000000983.png to avatar/avatar-000000983.png
  * updating image/avatar-000000984.png to avatar/avatar-000000984.png
  * updating image/avatar-000000986.png to avatar/avatar-000000986.png
  * updating image/avatar-000000987.png to avatar/avatar-000000987.png
  * updating image/avatar-000000989.png to avatar/avatar-000000989.png
  * updating image/avatar-000000991.png to avatar/avatar-000000991.png
  * updating image/avatar-000000992.png to avatar/avatar-000000992.png
  * updating image/avatar-000000993.png to avatar/avatar-000000993.png
  * updating image/avatar-000000995.png to avatar/avatar-000000995.png
  * updating image/avatar-000000997.png to avatar/avatar-000000997.png
DB batch done

The execution of update_db_batch took 0.01 seconds, avg 0.0 seconds per row
```

The verbose information is available both on screen and in the log file.

(but in the log file lines are prefixed with timestamps)

TLDR – simulated environment preparation tool



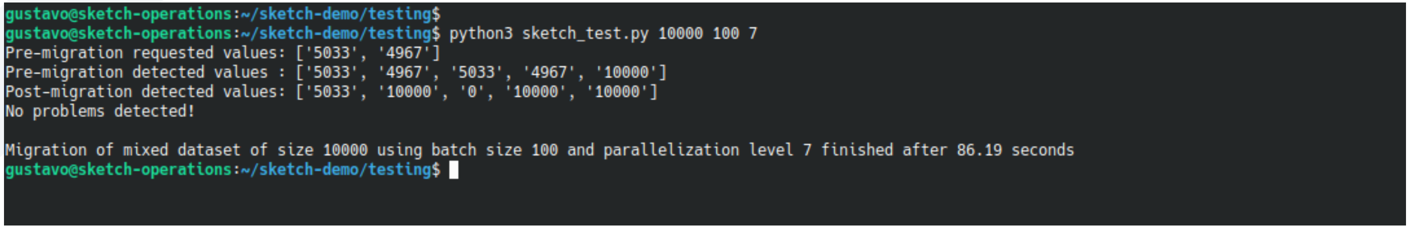
```
gustavo@sketch-operations:~/sketch-demo/preparation$ python3 sketch_prepare.py  -c 4000
WARNING: this script will DROP and re-CREATE database proddatabase and delete all contents of bucket sketch-legacy-s3 and bucket sketch-production-s3
Are you sure you want to continue? (yes/no) yes
Execution starting
Re-creating the database
Initializing the database
  * user migration created with password Ay6muiP9GuSl for migration purposes
Cleaning the legacy bucket
  * partial count: found 533 objects, deleted 533 objects
  * final count: found 533 objects, deleted 533 objects
Cleaning the production bucket
  * partial count: found 1000 objects, deleted 1000 objects
  * final count: found 1000 objects, deleted 1000 objects
Creating S3 objects and the corresponding database rows

Created 2001 legacy avatars and 1999 production avatars
gustavo@sketch-operations:~/sketch-demo/preparation$ 
```

Confirm you want reset the simulation data

Check progress

Review the final situation

TLDR – integration test



```
gustavo@sketch-operations:~/sketch-demo/testing$
gustavo@sketch-operations:~/sketch-demo/testing$ python3 sketch_test.py 10000 100 7
Pre-migration requested values: ['5033', '4967']
Pre-migration detected values : ['5033', '4967', '5033', '4967', '10000']
Post-migration detected values: ['5033', '10000', '0', '10000', '10000']
No problems detected!

Migration of mixed dataset of size 10000 using batch size 100 and parallelization level 7 finished after 86.19 seconds
gustavo@sketch-operations:~/sketch-demo/testing$ 
```

CAREFUL: this is an automated tool

Review correctness and performance

Compliance

Please check the compliance.md document for a one by one review of the requirements.

Agenda

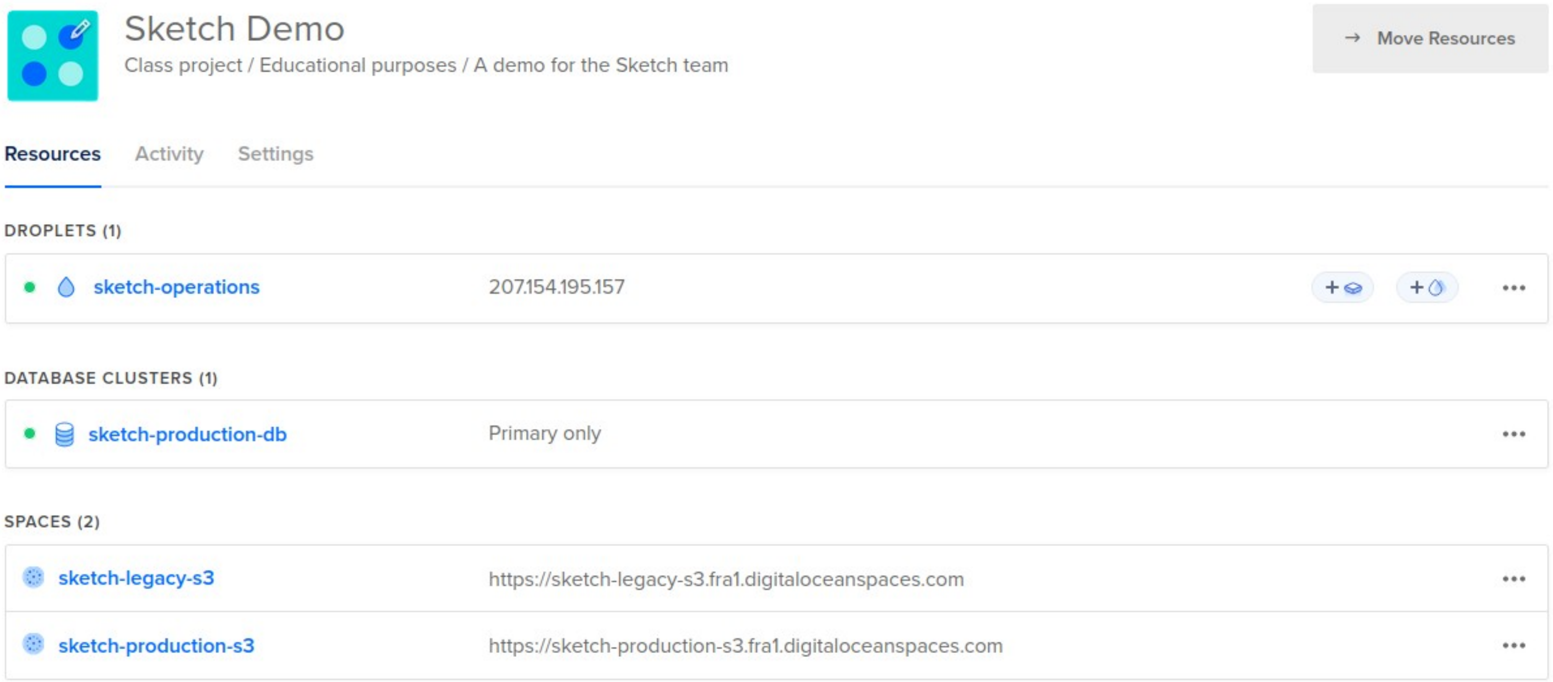
- **Environment**
- Deployment
- Performance

Environment

The execution environment is a Digital Ocean project comprised of:

- An Ubuntu Linux machine
- A PostgreSQL database cluster
- Two storage Spaces (S3 compatible buckets)

Environment



Sketch Demo

Class project / Educational purposes / A demo for the Sketch team

DROPLETS (1)

● sketch-operations 207.154.195.157

DATABASE CLUSTERS (1)

● sketch-production-db Primary only

SPACES (2)

sketch-legacy-s3 https://sketch-legacy-s3.fra1.digitaloceanspaces.com

sketch-production-s3 https://sketch-production-s3.fra1.digitaloceanspaces.com

Environment

I chose Digital Ocean because:

- the monthly Spaces subscription includes 1TB of bandwidth (performance testing generates many requests and AWS charges per request)
- performance results would not be realistic on local S3 emulation environments

Side effect: an opportunity to test the S3 API compatibility of a different provider.

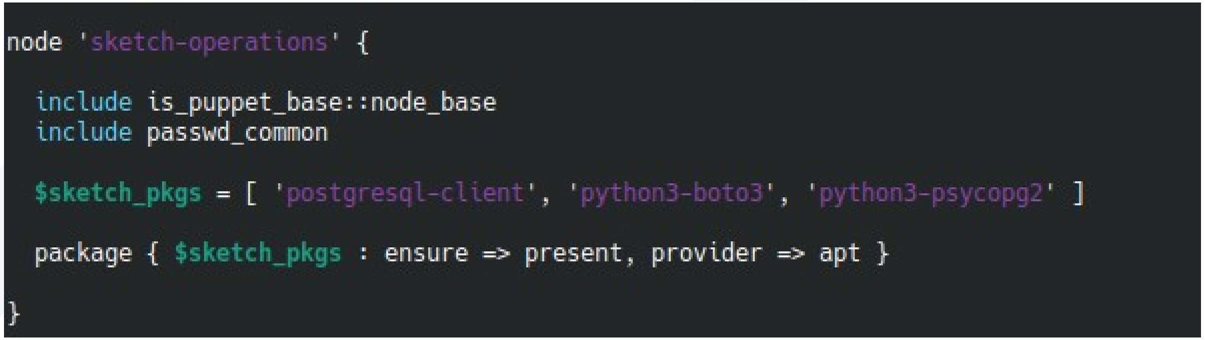
Agenda

- Environment
- **Deployment**
- Performance

Deployment

The deployment of the migration script is extremely simple because we only need an Ubuntu machine and the guarantee that 3 apt packages from the Ubuntu distribution are present.

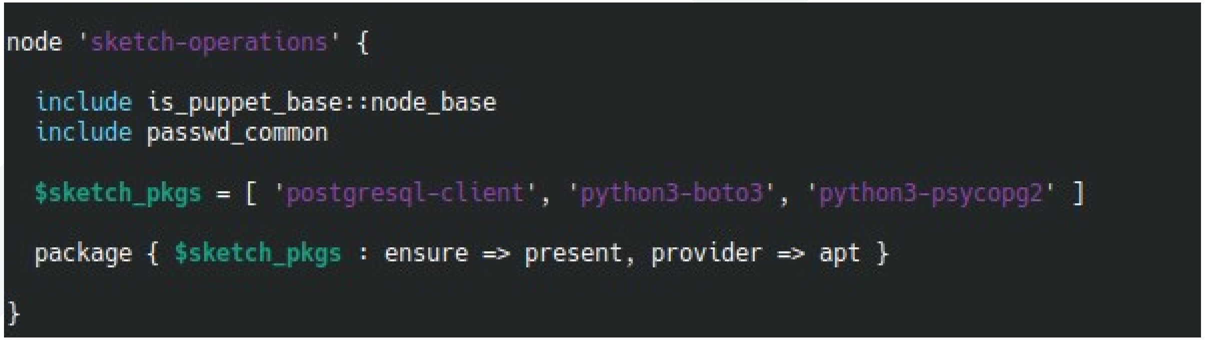
Deployment



```
node 'sketch-operations' {

  include is_puppet_base::node_base
  include passwd_common

  $sketch_pkgs = [ 'postgresql-client', 'python3-boto3', 'python3-psycopg2' ]

  package { $sketch_pkgs : ensure => present, provider => apt }

}
```

The machine

Deployment



```
node 'sketch-operations' {

  include is_puppet_base::node_base
  include passwd_common

  $sketch_pkgs = [ 'postgresql-client', 'python3-boto3', 'python3-psycopg2' ]

  package { $sketch_pkgs : ensure => present, provider => apt }

}
```

The machine

The deployment was tested in the smallest Digital Ocean configuration (1 vCPU, 512M RAM) and nothing bigger than that is necessary.

Deployment

The script could simply be deployed by hand using `git clone` or it could be deployed via IaC code. I can provide details on IaC deployment, if necessary.

Agenda

- Environment
- Deployment
- **Performance**

The low hanging fruit for performance

- Execute on a Linux instance near the database
- Avoid transversing the entire database more than once
- Do not overwrite files that are already migrated

The low hanging fruit for performance

- Execute on a Linux instance near the database
- Avoid transversing the entire database more than once
- Do not overwrite files that are already migrated

3x improvement versus “your computer” just from this, for single process executions.

The low hanging fruit for performance

- Execute on a Linux instance near the database
- Avoid transversing the entire database more than once
- Do not overwrite files that are already migrated

← 3x improvement versus "your computer" just from this, for single process executions.

The problem is very latency-sensitive because we are copying a large number of small files. For large files it would be different.

The low hanging fruit for performance

- Execute on a Linux instance near the database
- Avoid transversing the entire database more than once
- Do not overwrite files that are already migrated

3x improvement versus “your computer” just from this, for single process executions.

The problem is very latency-sensitive because we are copying a large number of small files. For large files it would be different.

It is not only client to API latency that counts. With small files there is also the TCP slow start inside the provider.

The low hanging fruit for performance

- Execute on a Linux instance near the database
- Avoid transversing the entire database more than once ← Theoretically important and adopted in the implementation but it turns out that the SELECTs with the LIKE operator are very fast compared to the S3 copy operations.
- Do not overwrite files that are already migrated

The low hanging fruit for performance

- Execute on a Linux instance near the database
- Avoid transversing the entire database more than once
- Do not overwrite files that are already migrated

From ~0.07 seconds per file to less than 0.005 seconds (more than 14x faster).

Note: some files might be already migrated if we are re-executing over a the result of previous partial migration

The low hanging fruit for performance

All the low hanging fruit has been collected before the implementation of more complex performance optimizations. A simple implementation with all the low hanging fruit included delivered a modest migration performance of ~ 14.5 entries per second.

Next steps for performance

With this implementation it was possible to conclude that:

- time spent on the single used SELECT is negligible
- time spent on UPDATEs is small compared to the total time
- **time spent on S3 copies dominates the process**

Next steps for performance

With this implementation it was possible to conclude that:

- time spent on the single used SELECT is negligible
- time spent on UPDATEs is small compared to the total time
- **time spent on S3 copies dominates the process**

We already lowered the latency by using a dedicated instance that lives near the database.

Next steps for performance

With this implementation it was possible to conclude that:

- time spent on the single used SELECT is negligible
- time spent on UPDATEs is small compared to the total time
- **time spent on S3 copies dominates the process**

← The S3 API does not implement bulk copies **in a simple way** ☹

Next steps for performance

With this implementation it was possible to conclude that:

- time spent on the single used SELECT is negligible
- time spent on UPDATEs is small compared to the total time
- **time spent on S3 copies dominates the process** ← I chose to parallelize the copies with the standard python multiprocessing module.

Strategy for multiprocessing

- scan the database and pick X batches of Y entries to migrate
  - distribute X batches to X processes and **wait for completion**
  - pick the execution results from the processes using a shared queue
  - log progress and stats

Strategy for multiprocessing

- scan the database and pick X batches of Y entries to migrate
  - distribute X batches to X processes and **wait for completion**
  - pick the execution results from the processes using a shared queue
  - log progress and stats

The waiting part might seem strange but we need to ensure that we don’t lose control of launched processes and we need to control the logging order of the migration steps. Furthermore, these processes will take roughly the same time to finish, therefore the waiting time of finished processes will be small compared to their execution time.

Strategy for multiprocessing

- scan the database and pick X batches of Y entries to migrate
  - distribute X batches to X processes and wait for completion
  - pick the execution results from the processes using a shared queue
  - log progress and stats
- implement an integration test that allows for detailed performance testing at low cognitive effort and find optimal values for X and Y

I executed the migration of 5000 entries 64 times on order to collect a modest set of performance data. If the process was not fully automated this would not have been possible within the given timeframe.

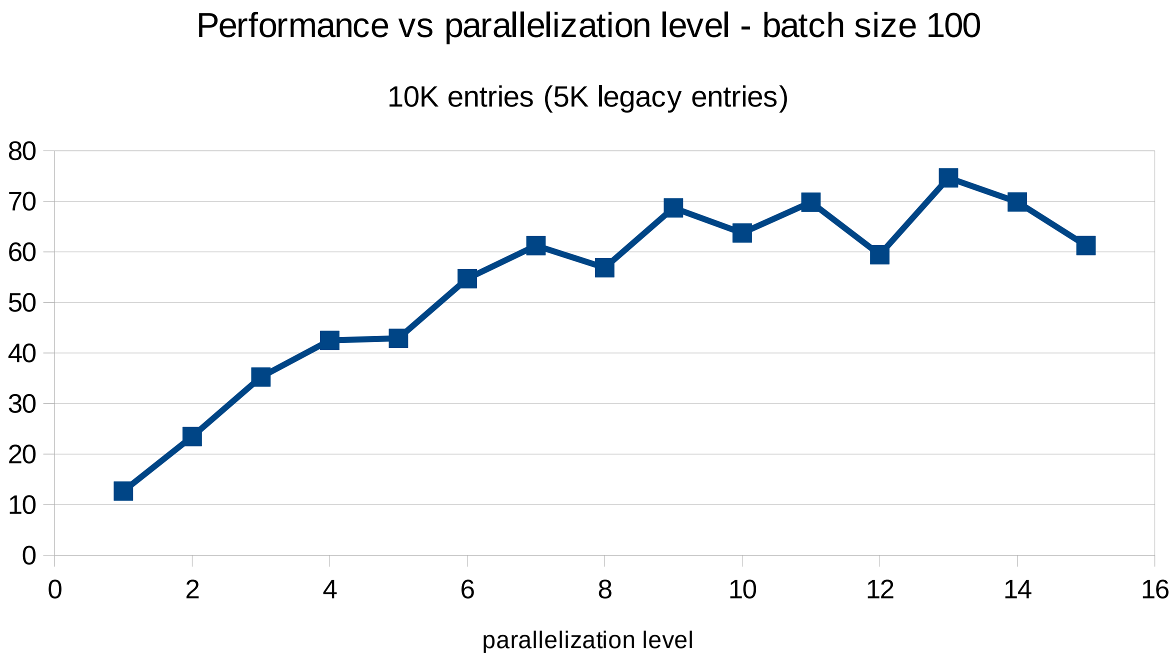
Strategy for multiprocessing

- scan the database and pick X batches of Y entries to migrate
  - distribute X batches to X processes and wait for completion
  - pick the execution results from the processes using a shared queue
  - log progress and stats
- implement an integration test that allows for detailed performance testing at low cognitive effort and find optimal values for X and Y

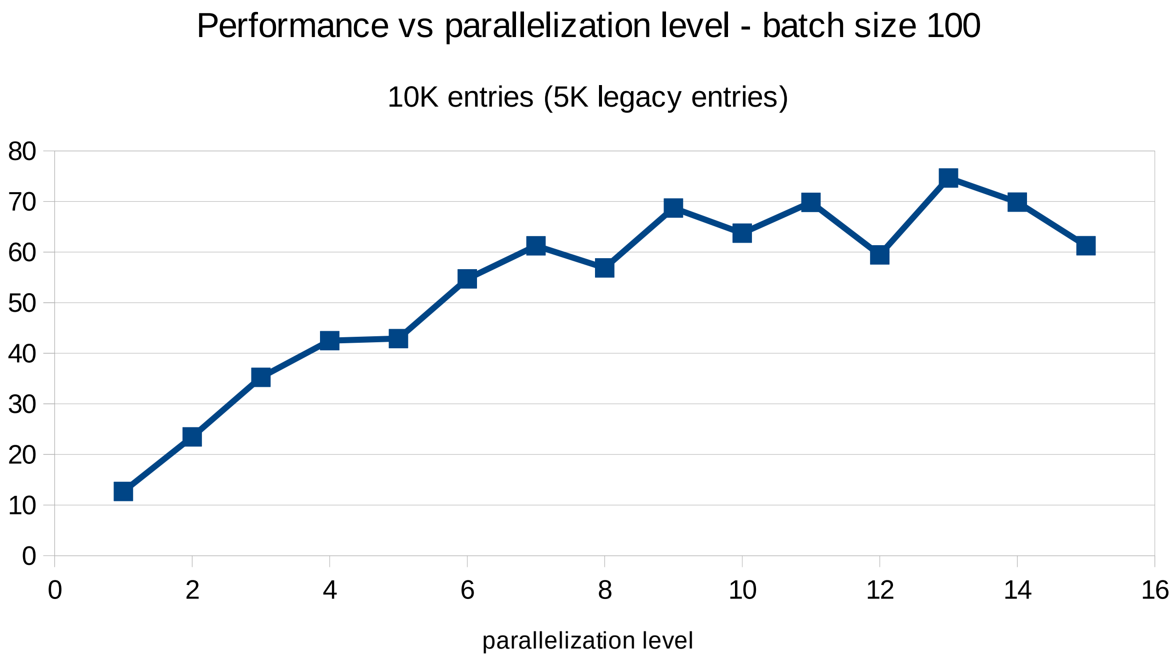
X and Y are exposed to the user as CLI parameters. See:

```
python3 sketch_migrate.py -h
```

Results obtained with multiprocessing

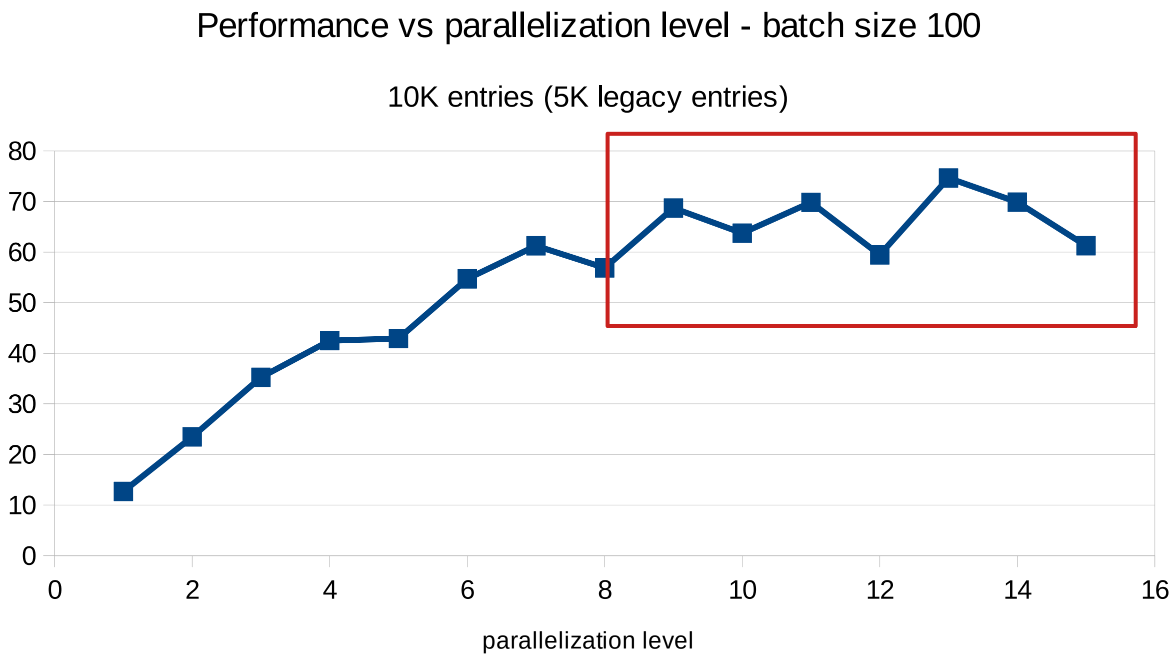


Results obtained with multiprocessing



As usual, real world data is noisy, but the pattern is clear.

Results obtained with multiprocessing



Performance *plateau*

Results obtained with multiprocessing

- 4x – 5x performance improvement
- the estimated performance is ~ 67 entries per second
- optimal batch size found to lie between 100 and 1000
- “optimal” parallelization level is between 5 and 10
  - there is a dramatic improvement as we increase the parallelization level from small numbers but a plateau is hit between 5 and 10

At 67 entries per second we would expect to migrate 1M entries in 4.15 h.

Tradeoffs

- No tradeoffs for the parallelization level: use at least of 10 processes as they fit perfectly even on a very small machine
- Small batches are less efficient because processes and network connections are started / stopped more often but large batches amplify the inter-batch waiting time for synchronization
- Small batches provide more frequent on screen progress updates than large batches
- **If in doubt, use a batch size of 100**

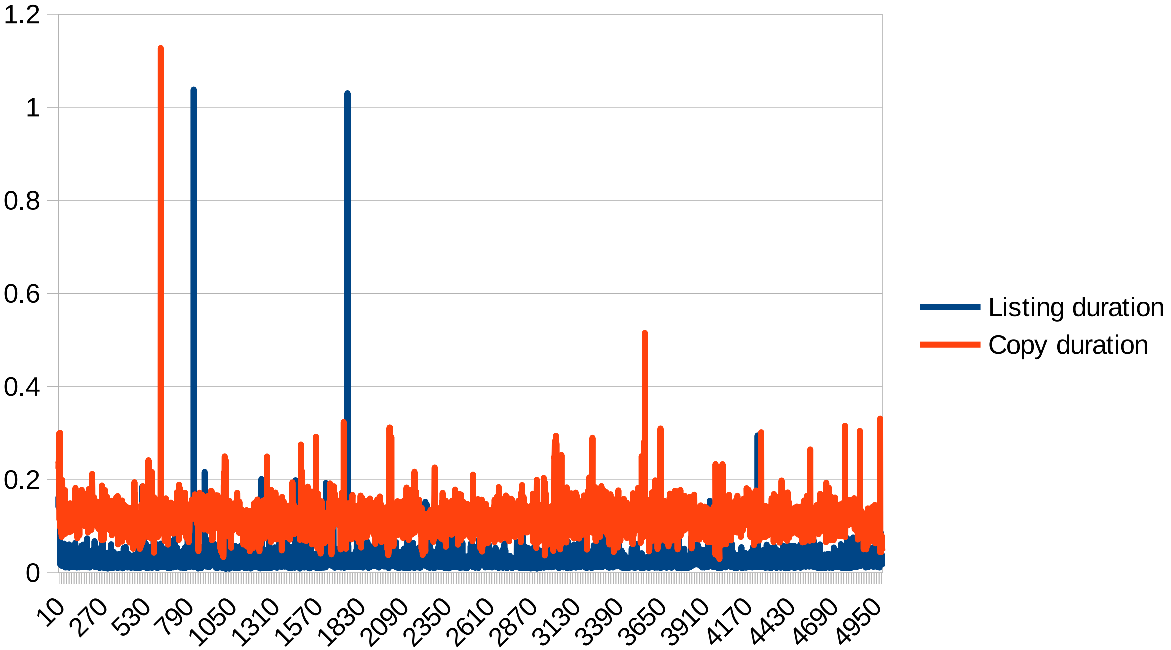
REVISITING the low hanging fruit for performance

- Execute on a Linux instance near the database
- Avoid transversing the entire database more than once
- **Do not overwrite files that are already migrated**

“From ~0.07 seconds per file to less than 0.005 seconds (more than 14x faster).”

Let’s measure this rigorously now!

The cost of overwrite prevention

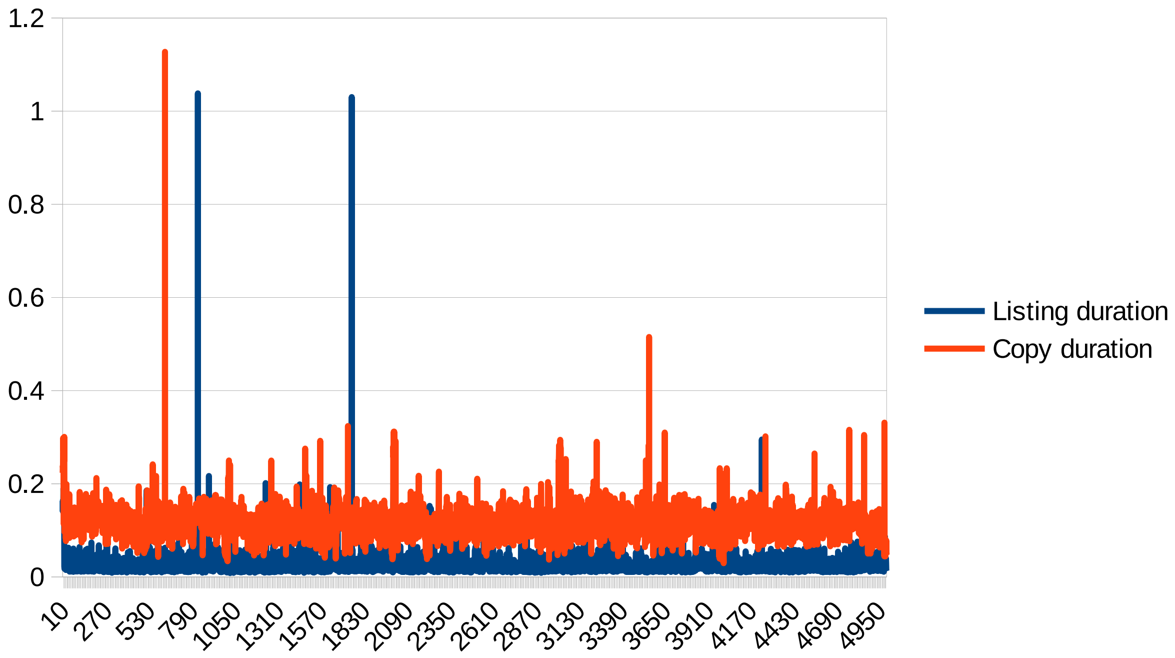


← The initial measurement was lighthearted and not accurate.

Copying takes in average 0.115 seconds while listing to look up the file takes in average 0.028 seconds.

(measured on a sample of 5000 points)

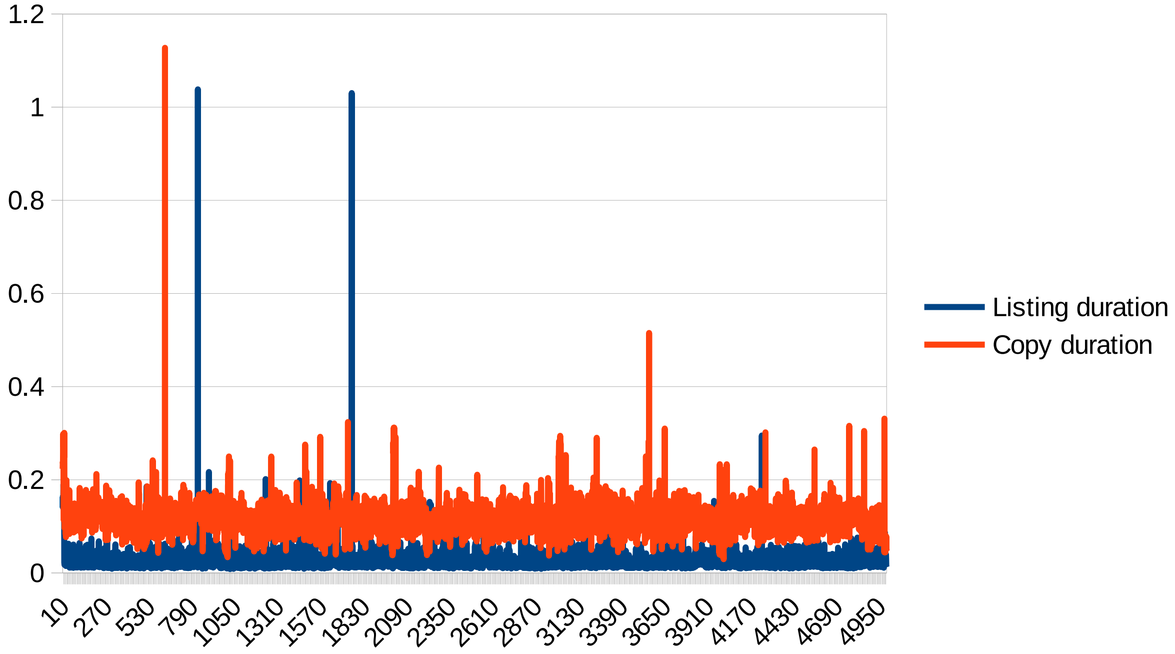
The cost of overwrite prevention



By looking up we use only 25% of what would be the copy time if the file is already in the production bucket (75% saved).

But when the file is not there, the lookup adds 25% to the copy process.

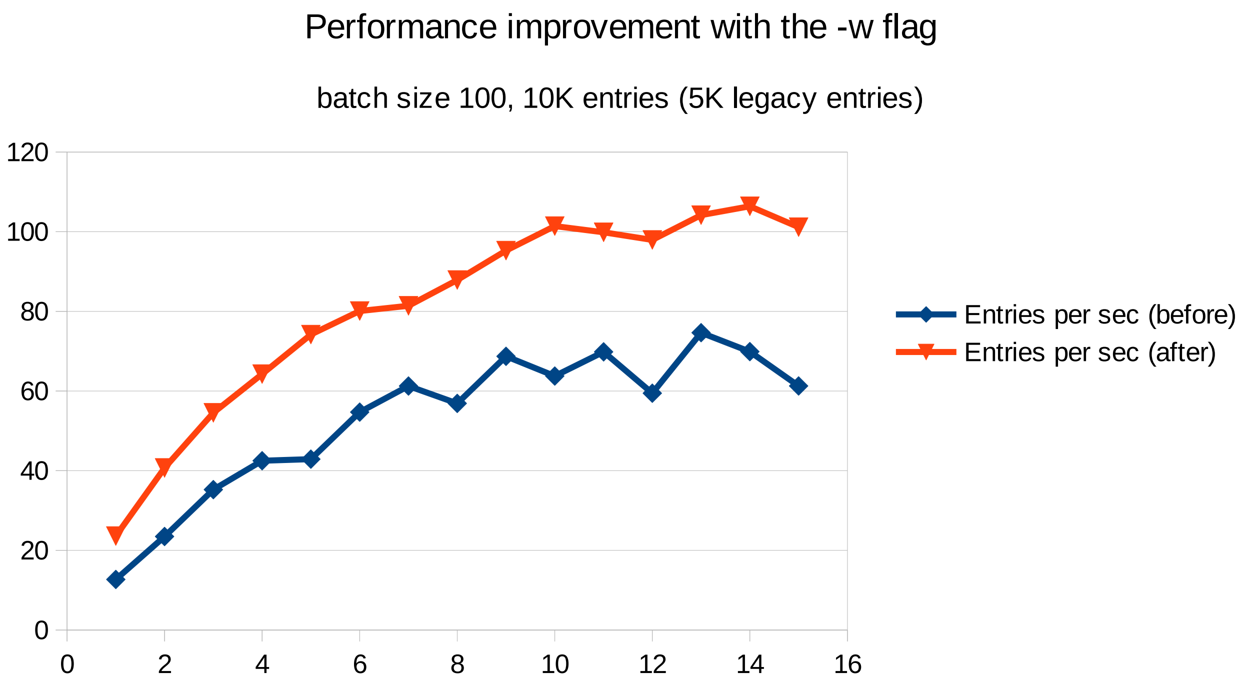
An idea for performance improvement



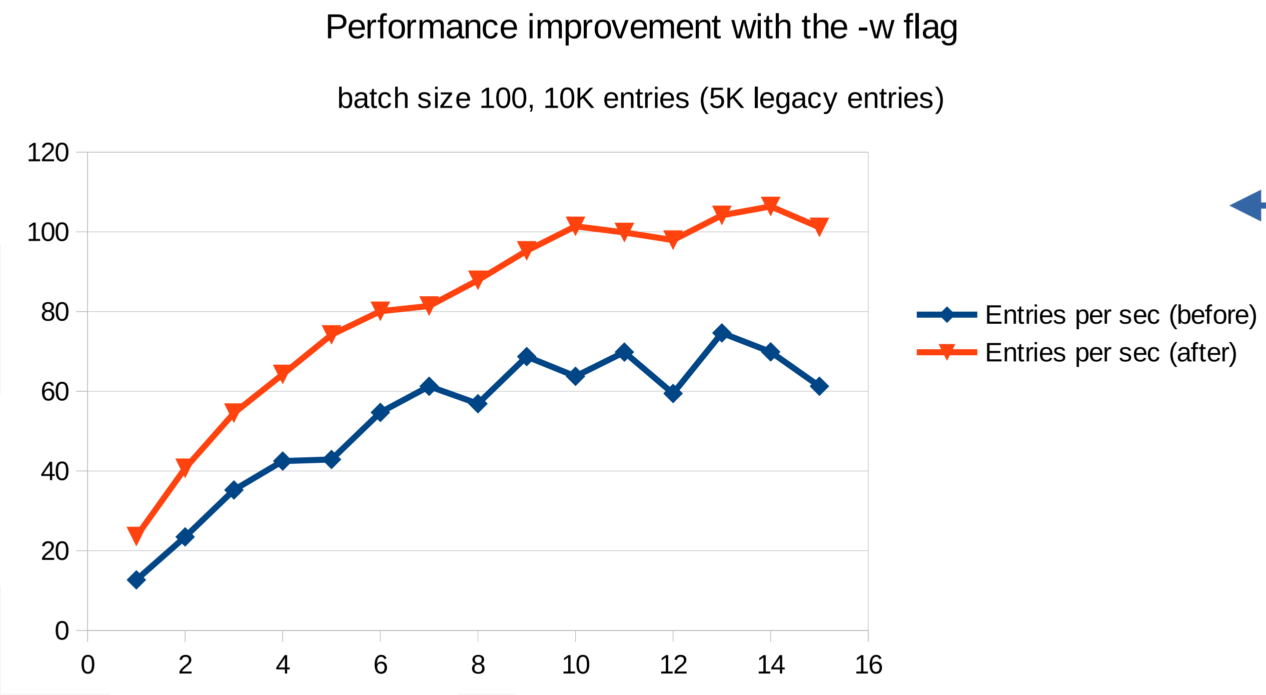
Idea: let's allow overriding the overwrite prevention via a new -w flag.

The prevention is not useful for the first execution, even though it might be useful for subsequent ones, if they are necessary.

Result


Performance improvement with the -w flag
batch size 100, 10K entries (5K legacy entries)
120
100
80
60
40
20
0
0
2
4
6
8
10
12
14
16
Entries per sec (before)
Entries per sec (after)

Result



The performance gain is ~ 50% rather than the expected 25%.

This could be because interleaving copy and list operations was causing an extra penalty compared to requesting copy operations only.

REVISITING results obtained with multiprocessing

- 7x performance improvement
- the estimated performance is ~ 100 entries per second
- optimal batch size found to lie between 100 and 1000 – **needs reconfirmation**
- “optimal” parallelization level is between 10 and 14

At 100 entries per second we would expect to migrate 1M entries in 2.8 h.

Are we happy, then?

Next Botoneck

Next Botoneck

Next Botoneck

Next Botoneck

`S3.Client.copy_object()`

is twice as fast as

`S3.Client.copy()`

at least for small files.

Next Botoneck

`S3.Client.copy_object()`

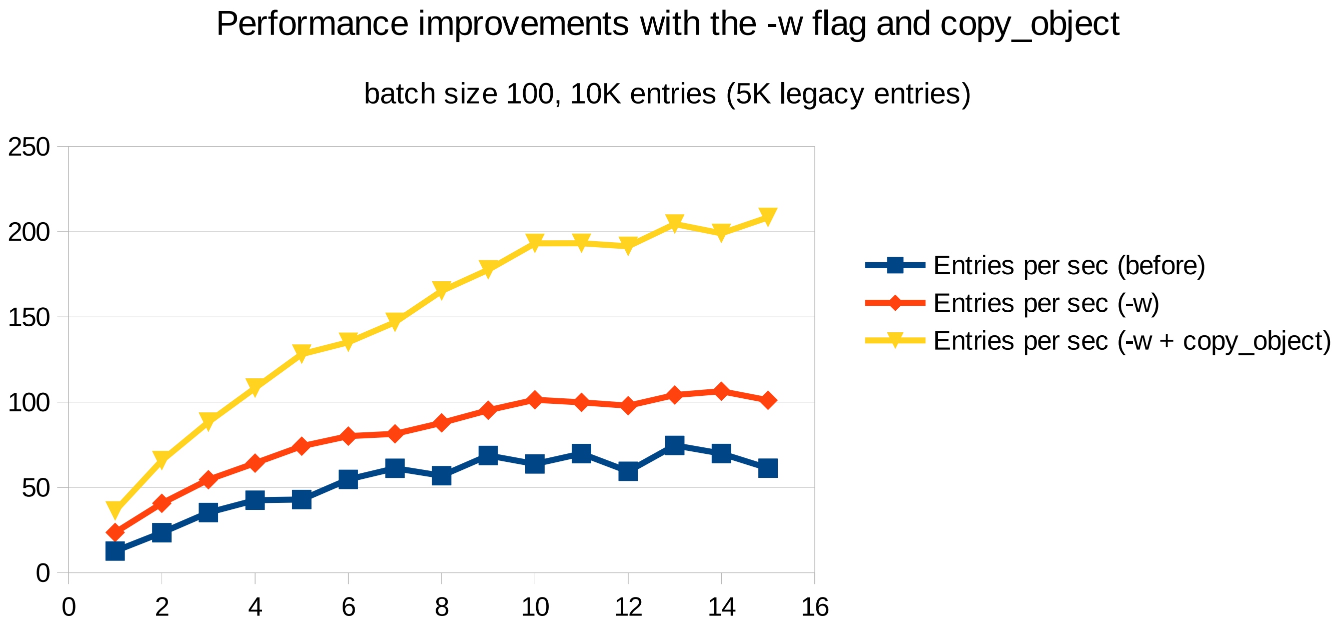
is twice as fast as

`S3.Client.copy()`

at least for small files.

Not entirely obvious detail!

Result


Performance improvements with the -w flag and copy_object
batch size 100, 10K entries (5K legacy entries)
250
200
150
100
50
0
0
2
4
6
8
10
12
14
16
Entries per sec (before)
Entries per sec (-w)
Entries per sec (-w + copy_object)

REVISITING results obtained with multiprocessing

- 14x performance improvement (vs the first implementation)
- the estimated performance is ~ 200 entries per second
- optimal batch size found to lie between 100 and 1000 – **needs reconfirmation**
- “optimal” parallelization level is between 10 and 15

At 200 entries per second we would expect to migrate 1M entries in 1.4 h.

Hypothetical other steps for performance

If this performance is not good enough we could still consider:

- refine the metrics (nr of observations and ranges)
- investigating the complex S3 Batch operations, as an alternative
- check if individual Linux instances are rate-limited in the S3 API and, if so, split the process across multiple instances

Each additional performance gain will come at the cost of additional R&D time and additional code complexity.

Final note

The performance results are orientative at this point. That is, these results allow us to understand the order of magnitude where we are working.

For the results to be statistically more robust we should perform multiple runs for every combination of batch size and parallelization level and calculate the average and standard deviation. There is intrinsinc variance in the process due to the variable load on the provider API.

The effort to invest in further performance work depends on how many millions of rows have to be migrated and how time sensitive is the process, given that there is no business discountinuity as the database entries are only updated after sucessful copy of the corresponding file to the new bucket.

Photo by Dave Hoefler on Unsplash

Thank you!

Sketch demo

Project highlights – rev 01

Gustavo Homem

homem.gustavo@gmail.com
October 2023