# Loadtest ANN query service with random embeddings

An ANN query service can be load-tested with random embeddings as queries, generated automatically by loadtest tool.

Example script to load test a ANN query service with random embeddings:

```bash

$ aurora job create smf1/<role>/staging/ann-loadtest-service ann/src/main/aurora/loadtest/loadtest.aurora \

--bind=profile.name=ann-loadtest-service \

--bind=profile.role=<role> \

--bind=profile.duration\_sec=10 \

--bind=profile.number\_of\_neighbors=10 \

--bind=profile.qps=200 \

--bind=profile.algo=hnsw \

--bind=profile.metric=Cosine \

--bind=profile.index\_id\_type=int \

--bind=profile.hnsw\_ef=400,600,800 \

--bind=profile.embedding\_dimension=3 \

--bind=profile.concurrency\_level=8 \

--bind=profile.loadtest\_type=remote \

--bind=profile.service\_destination=/srv#/staging/local/apoorvs/ann-server-test \

--bind=profile.with\_random\_queries=True \

--bind=profile.random\_queries\_count=50000 \

--bind=profile.random\_embedding\_min\_value=-10.0 \

--bind=profile.random\_embedding\_max\_value=10.0

```

It will run the loadtest with `50000` random embeddings, where each embedding value will be range bounded between `random\_embedding\_min\_value` and `random\_embedding\_max\_value`.

In the above the case it will be bounded between `-10.0` and `10.0`.

If `random\_embedding\_min\_value` and `random\_embedding\_max\_value` are not supplied default value of `-1.0` and `1.0` will be used.

## Results

Load test results will be printed to stdout of an aurora job.

# Loadtest ANN query service with query set

An ANN query service can be load-tested with sample queries drawn from the embeddings dataset.

For creating sample queries i.e `query\_set` refer this [section](#query-set-generator).

Test is run with `live` version of loadtest binary that is already available in packer.

Example script to load test a ANN query service:

```bash

$ aurora job create smf1/<role>/staging/ann-loadtest-service ann/src/main/aurora/loadtest/loadtest.aurora \

--bind=profile.name=ann-loadtest-service \

--bind=profile.role=<role> \

--bind=profile.duration\_sec=10 \

--bind=profile.query\_set\_dir=hdfs:///user/cortex/ann\_example/dataset/search/query\_knn/query\_set \

--bind=profile.number\_of\_neighbors=10 \

--bind=profile.qps=200 \

--bind=profile.algo=hnsw \

--bind=profile.query\_id\_type=string \

--bind=profile.index\_id\_type=string \

--bind=profile.metric=Cosine \

--bind=profile.hnsw\_ef=400,600,800 \

--bind=profile.embedding\_dimension=100 \

--bind=profile.concurrency\_level=8 \

--bind=profile.loadtest\_type=remote \

--bind=profile.service\_destination=/srv#/staging/local/apoorvs/ann-server-test

```

# In-Memory based loadtest for measuring recall

Load test can be with the above created dataset in memory.

For running in in-memory mode, index is created in memory, and for that you need `query\_set/index\_set/truth\_set`.

For creating this dataset refer this [section](#knn-truth-set-generator).

Test is run with `live` version loadtest binary that is already available in packer.

Example script In-Memory index building and benchmarking:

```bash

$ aurora job create smf1/<role>/staging/ann-loadtest ann/src/main/aurora/loadtest/loadtest.aurora \

--bind=profile.name=ann-loadtest \

--bind=profile.role=<role> \

--bind=profile.duration\_sec=10 \

--bind=profile.truth\_set\_dir=hdfs:///user/cortex/ann\_example/dataset/search/query\_knn/true\_knn \

--bind=profile.query\_set\_dir=hdfs:///user/cortex/ann\_example/dataset/search/query\_knn/query\_set \

--bind=profile.index\_set\_dir=hdfs:///user/cortex/ann\_example/dataset/search/query\_knn/index\_set \

--bind=profile.number\_of\_neighbors=10 \

--bind=profile.qps=200 \

--bind=profile.algo=hnsw \

--bind=profile.query\_id\_type=string \

--bind=profile.index\_id\_type=string \

--bind=profile.metric=Cosine \

--bind=profile.hnsw\_ef\_construction=15 \

--bind=profile.hnsw\_max\_m=10 \

--bind=profile.hnsw\_ef=400,600,800 \

--bind=profile.embedding\_dimension=100 \

--bind=profile.concurrency\_level=8 \

--bind=profile.loadtest\_type=local

```

# Loadtest faiss

```bash

$ aurora job create smf1/<role>/staging/ann-loadtest-service ann/src/main/aurora/loadtest/loadtest.aurora \

--bind=profile.name=ann-loadtest-service \

--bind=profile.role=<role> \

--bind=profile.duration\_sec=10 \

--bind=profile.number\_of\_neighbors=10 \

--bind=profile.qps=200 \

--bind=profile.algo=faiss \ # Changed to faiss

--bind=profile.faiss\_nprobe=1,3,9,27,81,128,256,512 \ # Added

--bind=profile.faiss\_quantizerKfactorRF=1,2 \ # Pass a list to do grid search

--bind=profile.faiss\_quantizerNprobe=128 \ # Added

--bind=profile.metric=Cosine \

--bind=profile.index\_id\_type=int \

--bind=profile.embedding\_dimension=3 \

--bind=profile.concurrency\_level=8 \

--bind=profile.loadtest\_type=remote \

--bind=profile.service\_destination=/srv#/staging/local/apoorvs/ann-server-test \

--bind=profile.with\_random\_queries=True \

--bind=profile.random\_queries\_count=50000 \

--bind=profile.random\_embedding\_min\_value=-10.0 \

--bind=profile.random\_embedding\_max\_value=10.0

```

Full list of faiss specific parameters. [Exact definition of all available parameters](https://github.com/facebookresearch/faiss/blob/36f2998a6469280cef3b0afcde2036935a29aa1f/faiss/AutoTune.cpp#L444). Please reach out if you need to use parameters which aren't shown below

```

faiss\_nprobe = Default(String, '1')

faiss\_quantizerEf = Default(String, '0')

faiss\_quantizerKfactorRF = Default(String, '0')

faiss\_quantizerNprobe = Default(String, '0')

faiss\_ht = Default(String, '0')

```

# Query Set Generator

Sample queries can be generated from the embeddings dataset and can be used directly with load test in tab format.

To generate sample queries `EmbeddingSamplingJob` can be used as follows.

```bash

$ ./bazel bundle cortex-core/entity-embeddings/src/scala/main/com/twitter/scalding/util/EmbeddingFormat:embeddingformat-deploy

$ export INPUT\_PATH=/user/cortex/embeddings/user/tfwproducersg/embedding\_datarecords\_on\_data/2018/05/01

$ export ENTITY\_KIND=user

$ export EMBEDDING\_INPUT\_FORMAT=usertensor

$ export OUTPUT\_PATH=/user/$USER/sample\_embeddings

$ export SAMPLE\_PERCENT=0.1

$ oscar hdfs \

--screen --tee log.txt \

--hadoop-client-memory 6000 \

--hadoop-properties "yarn.app.mapreduce.am.resource.mb=6000;yarn.app.mapreduce.am.command-opts='-Xmx7500m';mapreduce.map.memory.mb=7500;mapreduce.reduce.java.opts='-Xmx6000m';mapreduce.reduce.memory.mb=7500;mapred.task.timeout=36000000;" \

--min-split-size 284217728 \

--bundle embeddingformat-deploy \

--host hadoopnest1.smf1.twitter.com \

--tool com.twitter.scalding.entityembeddings.util.EmbeddingFormat.EmbeddingSamplingJob -- \

--entity\_kind $ENTITY\_KIND \

--input.embedding\_path $INPUT\_PATH \

--input.embedding\_format $EMBEDDING\_INPUT\_FORMAT \

--output.embedding\_path $OUTPUT\_PATH \

--output.embedding\_format tab \

--sample\_percent $SAMPLE\_PERCENT

```

It will sample 0.1% of embeddings and store them in `tab` format to hdfs that can be direcly used as `query\_set` for loadtest.

# Knn Truth Set Generator

To use load test framework to benchmark recall, you need to split your data set into index\_set, query\_set and knn\_truth

- index\_set: data that will be indexed for ann

- query\_set: data that will be used for queries

- truth\_set: the real nearest neighbor used as truth to compute recall

And also you need to figure out the dimension for your embedding vectors.

KnnTruthSetGenerator can help to prepare data sets:

```bash

$ ./bazel bundle ann/src/main/scala/com/twitter/ann/scalding/offline:ann-offline-deploy

$ export QUERY\_EMBEDDINGS\_PATH=/user/cortex-mlx/official\_examples/ann/non\_pii\_random\_user\_embeddings\_tab\_format

$ export INDEX\_EMBEDDINGS\_PATH=/user/cortex-mlx/official\_examples/ann/non\_pii\_random\_user\_embeddings\_tab\_format

$ export TRUTH\_SET\_PATH=/user/$USER/truth\_set

$ export INDEX\_SET\_PATH=/user/$USER/index\_set

$ export QUERY\_SET\_PATH=/user/$USER/query\_set

$ export METRIC=InnerProduct

$ export QUERY\_ENTITY\_KIND=user

$ export INDEX\_ENTITY\_KIND=user

$ export NEIGHBOURS=10

$ oscar hdfs \

--screen --tee log.txt \

--hadoop-client-memory 6000 \

--hadoop-properties "yarn.app.mapreduce.am.resource.mb=6000;yarn.app.mapreduce.am.command-opts='-Xmx7500m';mapreduce.map.memory.mb=7500;mapreduce.reduce.java.opts='-Xmx6000m';mapreduce.reduce.memory.mb=7500;mapred.task.timeout=36000000;" \

--bundle ann-offline-deploy \

--min-split-size 284217728 \

--host hadoopnest1.smf1.twitter.com \

--tool com.twitter.ann.scalding.offline.KnnTruthSetGenerator -- \

--neighbors $NEIGHBOURS \

--metric $METRIC \

--query\_entity\_kind $QUERY\_ENTITY\_KIND \

--query.embedding\_path $QUERY\_EMBEDDINGS\_PATH \

--query.embedding\_format tab \

--query\_sample\_percent 50.0 \

--index\_entity\_kind $INDEX\_ENTITY\_KIND \

--index.embedding\_path $INDEX\_EMBEDDINGS\_PATH \

--index.embedding\_format tab \

--index\_sample\_percent 90.0 \

--query\_set\_output.embedding\_path $QUERY\_SET\_PATH \

--query\_set\_output.embedding\_format tab \

--index\_set\_output.embedding\_path $INDEX\_SET\_PATH \

--index\_set\_output.embedding\_format tab \

--truth\_set\_output\_path $TRUTH\_SET\_PATH \

--reducers 100

```

It will sample 90% of index set embeddings and 50% of query embeddings from total and then it will generate 3 datasets from the same that are index set, query set and true nearest neighbours from query to index in the tab format.

`Note`: The reason for using high sample percent is due to the fact the sample embeddings dataset is small. For real use cases query set should be really small.

Set `--reducers` according to the embeddings dataset size.

# FAQ

There are multiple type of `query\_id\_type` and `index\_id\_type` that can be used. Some native types like string/int/long or related to entity embeddings

like tweet/word/user/url... for more info: [Link](https://cgit.twitter.biz/source/tree/src/scala/com/twitter/cortex/ml/embeddings/common/EntityKind.scala#n8)