Query-Based Simple and Scalable Recommender Systems with Apache Hivemall

Takuya Kitazawa
Treasure Data, Inc.
kitazawa@treasure-data.com

Makoto Yui
Treasure Data, Inc.
myui@treasure-data.com

ABSTRACT

This study demonstrates a way to build large-scale recommender systems by just writing a series of SQL-like queries. In order to efficiently run recommendation logics on a cluster of computers, we implemented a variety of recommendation algorithms and common recommendation functions (e.g., efficient similarity computation, top-k retrieval, and evaluation measures) as Hive user-defined functions (UDFs) in Apache Hivemall. We demonstrate that how Apache Hivemall can easily be used for building a scalable recommendation system with satisfying business requirements such as scalability, latency, and stability.

CCS CONCEPTS

• **Information systems → Recommender systems**; *Query languages*; MapReduce languages; Top-k retrieval in databases;

KEYWORDS

Hadoop; Hive; Spark; Factorization methods; Top-k item recommendation

ACM Reference Format:
Takuya Kitazawa and Makoto Yui. 2018. Query-Based Simple and Scalable Recommender Systems with Apache Hivemall. In *Twelfth ACM Conference on Recommender Systems (RecSys '18), October 2–7, 2018, Vancouver, BC, Canada.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3240323.3241592

1 INTRODUCTION

Real-world user behavioral data used for recommendation (e.g., e-commerce and online news) easily grows to more than dozens of gigabytes. Accordingly, Hadoop Distributed File Systems or Amazon S3 is often used for data lake where data is aggregated, and a certain number of developers are using Apache Hadoop and/or Apache Spark to achieve scalable implementation of their own custom recommendation methods with or without Spark MLlib and Apache Mahout. On that point, while there are convenient standalone recommendation libraries like LensKit [1], LibRec [3] and MyMediaLite [2] out there, their scalability is limited due to memory capacity of a single machine. Plus, even though user activity logs can frequently be stored as relational table forms, there is a lack of consideration for database-friendly optimizations such as joins and selections that are required for preparing inputs to recommender systems. In fact, Sarwat et al. [9] employed PostgreSQL to build their recommender, but this topic remains controversial.

This paper provides an alternative way for building a scalable recommender system using **Apache Hivemall**, a scalable machine learning and recommendation library that runs on Apache Hive and Spark [4]. We demonstrate a unique query-based recommender system which is particularly suitable for large-scale user-item interactions aggregated as tables on Apache Hive, a data warehouse environment built on the top of Apache Hadoop. Apache Hivemall enables us to easily implement various recommendation logics from well-studied techniques to up-to-date algorithms in a scalable manner just by issuing series of HiveQL queries.

2 APACHE HIVEMALL

In addition to recommendation algorithms, Apache Hivemall has a variety of functionalities including regression, classification, anomaly detection, TF-IDF computation, top-k data processing, and natural language processing; some of them are particularly useful for building a recommendation engine. As illustrated in Figure 1, Hivemall is built on the Hadoop ecosystem so that it can leverage the computational efficiency of Hadoop for large-scale data processing. Notice that Hivemall is flexible at the choice of each layer; it can use Apache Tez, Apache Spark, or the plain old MapReduce runtime for parallel data processing.

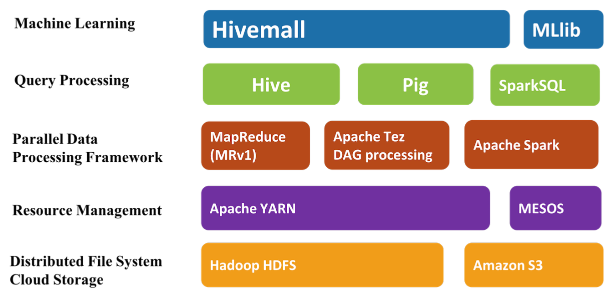

Figure 1: Technical components surrounding Hivemall.

Installation of Hivemall to an existing Hadoop/Hive environment (e.g., Amazon EMR) is straightforward as follows[1]:

```
hive> add jar /path/to/hivemall-all-(VERSION).jar;
hive> source /path/to/define-all.hive;
```

3 RECOMMENDATION WITH HIVEMALL

Users can enjoy a wealth of recommendation algorithms in Hivemall. For instance, it is possible to apply classical collaborative filtering techniques for user-item transactional data by using a UDF `cosine_similarity`[2]. Meanwhile, Hivemall supports state-of-the-art efficient alternatives like SLIM [6] and DIMSUM [10] as well as well-known factorization methods such as Matrix Factorization (MF) and (Field-Aware) Factorization Machines [5, 7].

[1] Find the details in our user guide: http://hivemall.incubator.apache.org/userguide/
[2] http://hivemall.incubator.apache.org/userguide/recommend/item_based_cf.html

3.1 Example: Explicit Rating Prediction

From a practical point of view, the fact that all of the techniques can be implemented in small query fragments significantly simplifies day-to-day recommendation workflows; we can build a MF model by just issuing the 14-lines query as seen in List 1. In List 1, a source table `train` has `<userid,itemid,rating>` for each row, and Hivemall function `train_mf_sgd` learns latent factors with a set of hyper-parameters passed to the fourth argument. `GROUP BY idx` aggregates and averages intermediate results computed in parallel by different workers.

List 1: Build MF model from rating data.

```sql
CREATE TABLE mf_model AS
SELECT
  idx,
  array_avg(u_rank) AS Pu, array_avg(i_rank) AS Qi,
  AVG(u_bias) AS Bu, AVG(i_bias) AS Bi -- bias terms
FROM (
  SELECT
    train_mf_sgd(
      userid, itemid, rating,
      '-factor 4 -lambda 0.001 -eta 0.1'
    ) AS (idx, u_rank, i_rank, u_bias, i_bias)
  FROM train
) t
GROUP BY idx;
```

Once a MF model has been created, the model table `mf_model` can be utilized as List 2 to predict unforeseen ratings. Since Hivemall stores both user- and item-side model parameters into single table, `JOIN`-ing the table twice with `test` table allows us to find corresponding parameters to each target user-item pair. Eventually, a UDF `mf_predict` internally calculates predicted value as: $B_u + B_i + P_u^\top Q_i$, by letting B_u, P_u and B_i, Q_i be a pair of bias term and latent vector for user u and item i, respectively. Additionally, List 3 measures the accuracy of prediction by using `rmse` (Root Mean Squared Error) and `mae` (Mean Absolute Error) UDFs.

List 2: Rating prediction based on the MF model.

```sql
CREATE TABLE predictions AS
SELECT
  t2.userid, t2.itemid,
  mf_predict(t2.Pu, m2.Qi, t2.Bu, m2.Bi) AS rating
FROM ( -- user-side model parameters
  SELECT
    t1.userid, t1.itemid, m1.Pu, m1.Bu
  FROM
    test t1 LEFT OUTER JOIN mf_model m1
    ON (t1.userid = m1.idx)
) t2
-- join with item-side parameters
LEFT OUTER JOIN mf_model m2 ON (t2.itemid = m2.idx);
```

List 3: Evaluate the accuracy of prediction.

```sql
SELECT
  mae(p.rating, t.rating) AS mae,
  rmse(p.rating, t.rating) AS rmse
FROM
  predictions p JOIN test t
  ON (p.userid = t.userid AND p.itemid = t.itemid);
```

3.2 Top-K Item Recommendation

As a result of rating prediction, production systems often offer recommending top-k items for each user. In order to cope with the common scenario, Hivemall provides `each_top_k` UDF that efficiently runs top-k (tail-k) retrieval. The UDF internally holds a bounded priority queue to sort elements by a specific key with less memory consumption, rather than keeping whole items. Thanks to the highly optimized implementation, `each_top_k` readily finds top-k `itemid` for each `userid`. Even though Hive native query (i.e., combination of `rank() over` and `ORDER BY`) takes dozens of hours to find top-k items on millions of records, Hivemall `each_top_k` UDF generally finishes the same task in a few hours.

List 4: Top-3 recommendation based on predicted ratings.

```sql
WITH user_model AS ( -- all user-side model parameters
  SELECT idx AS userid, Pu, Bu FROM mf_model m
  WHERE m.idx IN (SELECT userid FROM users)
),
item_model AS ( -- all item-side model parameters
  SELECT idx AS itemid, Qi, Bi FROM mf_model m
  WHERE m.idx IN (SELECT itemid FROM items)
)
SELECT
  -- top-3 items for each user based on MF prediction
  each_top_k(
    3, u.userid, mf_predict(u.Pu, i.Qi, u.Bu, i.Bi),
    u.userid, itemid
  ) AS (rank, score, userid, itemid)
FROM user_model u
LEFT OUTER JOIN item_model i
WHERE NOT EXISTS ( -- ignore user-item pairs in train set
  SELECT userid FROM train t
  WHERE t.userid = u.userid AND t.itemid = i.itemid
);
```

4 CONCLUSION

We have demonstrated a new type of large-scale recommendation engine built on the top of Hadoop ecosystem using Apache Hivemall. The recommender fully leverages Hadoop's scalability and fault tolerance, and HiveQL's simplicity and flexibility. While this paper focused on simple recommendation examples for explicit feedback datasets, Hivemall can also be applied for implicit feedback datasets by using BPR-MF [8]. Moreover, other Hivemall capabilities such as approximate similarity computation (e.g., MinHash and DIMSUM [10]), efficient top-k recommendation [6], feature-based recommendation [5, 7], and evaluation using various accuracy measures play a crucial role to make our recommender more feasible. Our demo session auxiliary shows several real-world examples using Apache Hivemall in production environment.

A LINK TO NARRATED SCREEN CAPTURE

https://www.youtube.com/watch?v=cMUsuA9KZ_c

REFERENCES

[1] M. D. Ekstrand et al. 2011. Rethinking the Recommender Research Ecosystem: Reproducibility, Openness, and LensKit. In *Proc. RecSys 2011*. 133–140.
[2] Z. Gantner et al. 2011. MyMediaLite: A Free Recommender System Library. In *Proc. RecSys 2011*. 305–308.
[3] G. Guo et al. 2015. LibRec: A Java Library for Recommender Systems. In *Proc. UMAP 2015*.
[4] Apache Incubator. 2016. Apache Hivemall. http://hivemall.incubator.apache.org/. (2016). (visited on July 2, 2018).
[5] Y. Juan et al. 2016. Field-Aware Factorization Machines for CTR Prediction. In *Proc. RecSys 2016*. ACM, 43–50.
[6] X. Ning and G. Karypis. 2011. SLIM: Sparse Linear Methods for Top-N Recommender Systems. In *Proc. ICDM 2011*. 497–506.
[7] S. Rendle. 2012. Factorization Machines with libFM. *ACM TIST* 3, 3 (May 2012).
[8] S. Rendle et al. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proc. UAI 2009*. 452–461.
[9] M. Sarwat et al. 2013. RecDB in Action: Recommendation Made Easy in Relational Databases. *Proc. VLDB Endowment* 6, 12 (2013), 1242–1245.
[10] R. B. Zadeh and G. Carlsson. 2013. Dimension Independent Matrix Square using MapReduce (poster). In *Proc. STOC 2013*.