A Hybrid Approach to Social Network User Feed Generation

Akhil Sudhakaran
Department of Computer Science,
Sir M. Visvesvaraya Institute of Technology,
Bengaluru, Karnataka, India

Devipriya Sarkar
Department of Computer Science,
Sir M. Visvesvaraya Institute of Technology,
Bengaluru, Karnataka, India

Praveen Kumar G.
Department of Computer Science,
Sir M. Visvesvaraya Institute of Technology,
Bengaluru, Karnataka, India

Ravikiran R.
Department of Computer Science,
Sir M. Visvesvaraya Institute of Technology,
Bengaluru, Karnataka, India

Sushila Shidnal
Asst. Prof., Department of Computer Science,
Sir M. Visvesvaraya Institute of Technology,
Bengaluru, Karnataka, India

ABSTRACT

Existing social networks handle generation of user activity feeds by utilizing different data distribution models. Different models optimize different aspects of feed generation such as user specificity, processing efficiency, resource utilization and latency. This paper propo*ses a hybrid model to handle this problem elegantly. This model takes into account the frequency of query requests between individual users and classifies them into either a PUSH-Target user or PULL-Target user. The former is provided with prioritized data pushes and the latter with data pulls on user request basis.

Keywords

Social Network, Activity Feed, Hybrid Model.

1. INTRODUCTION

A ‘feed’ is a content representation format used for providing users(subscribers) with frequently updated information. Feeds are populated with content provided by publishers that the user is subscribed to. This can be sorted based on the time of release or relevance to the user.

The steps involved in generating a unique feed for every user is decided based on the content that is served at the end. Several questions need to be answered to decide the technique required such as:

(1). Are there more producers than consumers?

(2). What is the maximum acceptable latency?

(3). Will the feed items be sorted in the chronological order or ranked selectively?

(4). Will the feed be infinitely long or of fixed length?

(5). Whether or not ads are a part of the feed to monetize the feed service, etc.

Based on these questions one needs to choose a strategy for publishing the feed.

2. RELATED WORK

2.1 Existing Models

There are primarily two models for managing feed events:

(1) Push Model / Fan-out-on-write (As shown in figure 1) - Each activity is pushed to a generated feed maintained for every consumer.

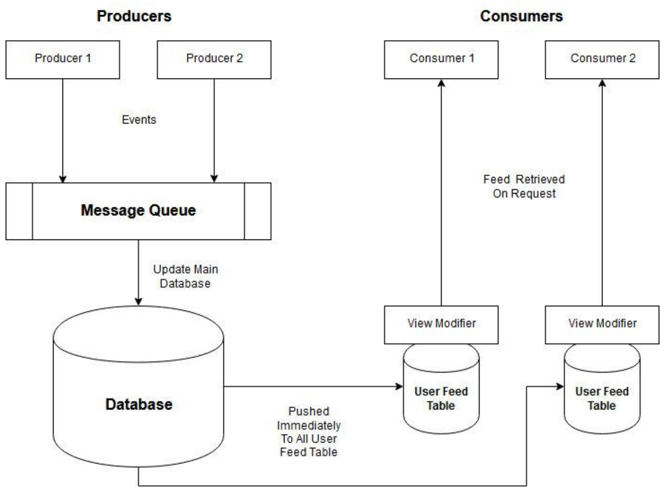

Fig. 1. Diagrammatic Representation of Push Model

(2) Pull Model / Fan-out-on-load (as shown in figure 2)- Activities are retrieved from producers when user logs in or refreshes the feed.

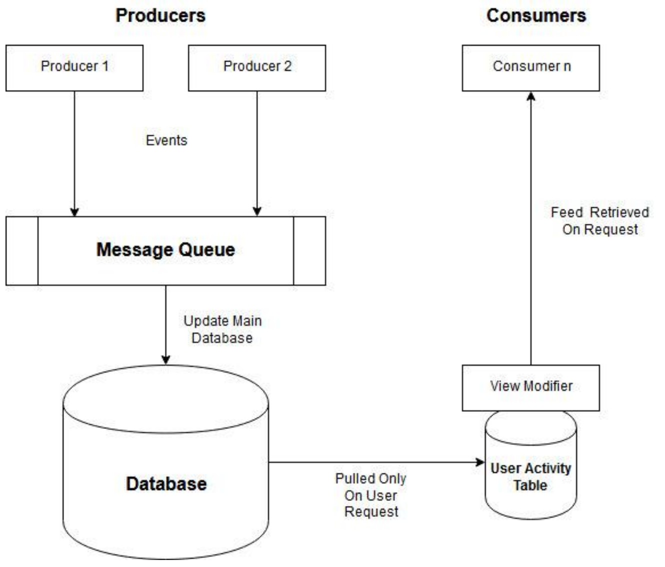

Fig. 2. Diagrammatic Representation of Pull Model

Using the push model results in lower latency when fetching feeds because they are pre-generated. This involves a huge number of writes, but reads are really fast. This can lead to higher resource consumption and potentially wasteful if users log in infrequently. In contrast, using the pull model requires processing time to generate a feed but is much more resource friendly.

Many of the popular platforms today use a combination of both. Although this leads to better performance, it significantly increases complexity of the architecture

2.2 Existing Implementations

2.2.1 Facebook's News Feed

Facebook's architecture uses a model that is primarily pull based [2]. News feeds are fetched using aggregators, which are query engines that accept user queries and retrieve items from the backend while performing aggregation, ranking [7] and filtering. Aggregators use 'leaf' servers, which is a distributed storage layer that indexes most recent activity for each user.

2.2.2 Twitter's Feed

Twitter uses a combination of both, pushing only to users that are currently active [1]. Fan outs are implemented as micro services. Cache is critical for Twitter and protects backing stores from heavy read traffic. It uses Manhattan, Blobstore, FlockDB and Redis among others.

2.2.3 Pinterest Feed

An implementation of the feed at Pinterest needs to be continually updated as users follow or unfollow other users and boards. New content is pushed out to the feeds of all followers. MySQL is used for persistence and HBase for backend storage [4].

2.2.4 Instagram Feed

In contrast to Facebook, Instagram uses a predominantly push based model for generating a user's feed. When a user uploads a post, the activity is asynchronously pushed to each of the user's followers using a task manager and a message broker [5]. Cassandra, Redis, RabbitMQ, PostgreSQL and Memcached are used to manage data [3].

2.2.5 Yahoo

Yahoo describes architectures and techniques for large scale applications that require the generation of activity feeds in its paper "Feeding Frenzy: Selectively Materializing Users' Event Feeds" [6]. It suggests selecting a push or pull action for each producer consumer pair based on their relative producing and querying frequencies and the cost of each action. This strategy was tested using Yahoo!'s web-scale database PNUTS [8], resulting in lowest system load.

3. PROPOSED MODEL

An optimal system utilization can be achieved by making local push/pull decisions on a per producer-consumer pair basis, depending on the relative frequencies of publishing and querying. This paper suggests a less complex alternative by associating the push/pull decision with each user.

Here, individual users are classified as either PUSH-target or PULL-target based on the frequency of querying for feed. If a user queries for updates frequently, he/she's classified as a PUSH-target. If a user queries relatively less, he/she's a PULL-target. The threshold frequency to decide whether a user is push or pull target is based on the average query frequency of all users and current system performance.

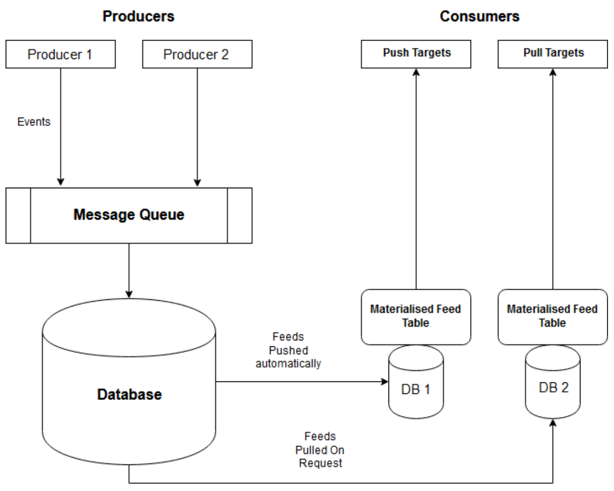

Fig. 3. Diagrammatic Representation of Hybrid Model

If a user has query frequency below the threshold, he/she is a PULL-target; else push. When an event is generated, it is only pushed to all the followers of the producers that are PUSH-targets. Thus when a user requests for a feed, the system first checks to see what kind of user it is. In case of a PUSH-target, the feed is directly fetched from that user's materialized feed table. In the other case, the feed is generated by fetching the activity of the producers in that user's network (As shown in figure 3).

4. IMPLEMENTATION

The abstract implementation of the push algorithm is stated below:

Algorithm 1 Algorithm for Push
Input: *e (Event), User*
Output: *User Feed*

```
1:  onEvent(e, User):
2:      for each follower of User:
3:          follower.Feed.push(e)
4:
5:  fetchFeed(User):
6:      return User.Feed
```

Each event by a producer is pushed to the feed table of all the subscribers of the producer. When a feed is requested by a user, it is fetched from the per-consumer materialized feed table. Ranking of items in the feed can be done either when the feed is fetched, or when an event is being pushed.

The abstract implementation of the pull algorithm is stated below:

Algorithm 2 Algorithm for Pull
Input: *e (Event), User*
Output: *Feed*

```
1:  onEvent(e, User):
2:      User.activity.push(e)
3:
4:  fetchFeed(User):
```

```
5:      Feed = []
6:      for each following of User:
7:          Feed.append(following.activity.top)
8:      return Feed
```

Each event by a producer is stored in the activity table of that user. When a feed is requested by a user, the algorithm fetches the top ‘x’ entries of all the producers the user is subscribed to. The value of ‘x’ determines the diversity level of the feed, to ensure items in the feed include events from multiple sources.

In the proposed implementation the user base is divided into a push-list and a pull-list. The deciding factor for this division is the frequency at which each user queries for the feed. This frequency is compared with a global average. If the requesting user's query frequency is lesser, then the user is categorized as a pull user, else the user becomes a push user.

The hybrid feed generation algorithm implementation is stated below:

Algorithm 3 Hybrid algorithm for feed generation

Input: *e (Event), User*

Output: *User Feed*

```
1:    on Event(e, User):
2:        for each PUSH follower of User
3:            follower.feed.push(e)
4:
5:    fetchFeed(User):
6:        if User is PUSH:
7:            return User.Feed
8:        else if User is PULL
9:            User.Feed = []
10:           for each following of User:
11:               User.Feed.append(following.activity.top)
12:       return User.Feed
```

In the hybrid algorithm, when an event is created it is still pushed to the subscribers of that user. But here, the push action is performed only to the subscribers categorized as PUSH-target. When a user requests for a feed, there are two ways it could be fetched depending on the category of the user:

(i). The user is a PUSH-target: In this case all the required events are present right in the feed table of the user. They are ranked, if required and presented to the user.

(ii). The user is a PULL-target: In this case, events need to be fetched from the producers the user is subscribed to. This happens in the same way as the pull model.

5. CONCLUSION

In this paper the proposed algorithm handles generation of personalized user activity feed. The goal has been to improve performance and resource utilization by means of a hybrid model while retaining simplicity of architecture. Classification of users is done based on their relative query frequency. The claim is that the average query rate is a good heuristic to discriminate users and prioritizing feed performance of active users leads to a fairly balanced trade-off between resource usage, performance and user experience.

6. FUTURE WORK

The paper hasn’t addressed the exact manner in which the threshold frequency is determined, and has assumed that this is the average query frequency of all users in the platform. Some strategies to determine this value include maintaining a service that tracks global average frequency of queries, and running periodic checks on query logs to compute the latest value. In the latter case the time period must vary as the system scales, to keep up with changes in user activity patterns. Different average values can be maintained for different geographical regions if required.

Another consideration is what happens when the category of a user changes. By default, a user is a PULL-target. On transitioning to a PUSH-target, feed will be pre-generated for that user. In the other transition case, the generated feed items of the user can either be discarded, or retained to quickly display an old feed while the latest one is being generated.

7. REFERENCES

[1] M. Hashemi “The Infrastructure Behind Twitter: Scale”, retrieved from https://blog.twitter.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale.html (2017, 19 January).

[2] Y. Zhu “Serving Facebook Multifeed: Efficiency, performance gains through redesign”, retrieved from https://code.facebook.com/posts/781984911887151/serving-facebook-multifeed-efficiency-performance-gains-through-redesign/ (2015, 10 March).

[3] S. Schneider “How Instagram Feed Works: Celery and RabbitMQ”, retrieved from https://blogs.vmware.com/vfabric/2013/04/how-instagram-feeds-work-celery-and-rabbitmq.html (2013, 15 April).

[4] V. Sharma *et al.* “Scaling Deep Social Feeds at Pinterest” *SocialCom*, 2013.

[5] Instagram Engineering “What Powers Instagram: Hundreds of Instances, Dozens of Technologies”, retrieved from https://instagram-engineering.com/what-powers-instagram-hundreds-of-instances-dozens-of-technologies-adf2e22da2ad (2011, 2 December).

[6] A. Silberstein *et al.* “Feeding Frenzy: Selectively Materializing Users’ Event Feeds” *SIGMOD*, June 6–11, 2010.

[7] M. Zuckerberg *et al.* "Dynamically providing a news feed about a user of a social network." U.S. Patent 7,669,123, issued February 23, 2010.

[8] B.F. Cooper *et al.* "PNUTS: Yahoo!'s hosted data serving platform." *Proceedings of the VLDB Endowment* 1, no. 2 2008: 1277-1288.