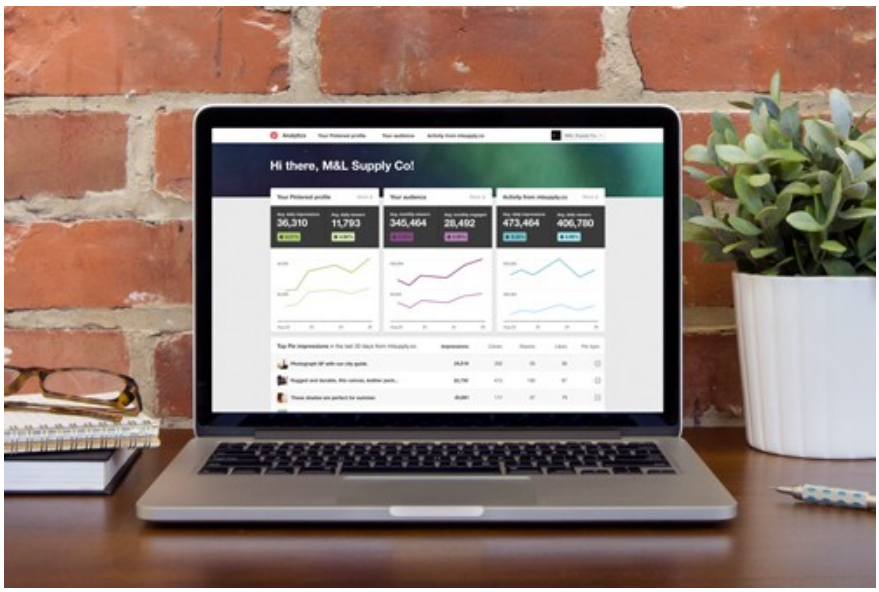
Pinterest Engineering Following
Inventive engineers building the first visual discovery engine, 100 billion ideas and counting. http...
Aug 29, 2014 · 6 min read

Behind the Pins: Building Analytics

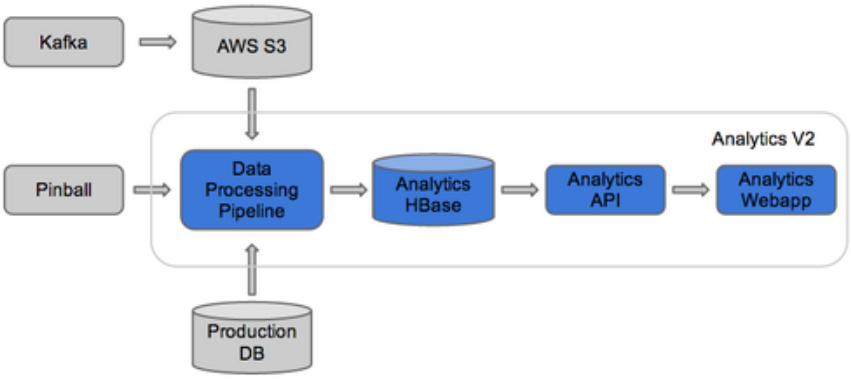
Tongbo Huang | Pinterest engineer, Ads

As the community of Pinners grows, so does the number of businesses on Pinterest. Earlier this week, we revamped our analytics tool to help businesses better understand their audiences and how their organic content is performing.



This second version of our analytics product offers new features such as profile and audience analytics, user country and metro segmentation, app segmentation, more comprehensive event types (such as impressions, clicks, and repins), and Pin It button analysis for the off-network and on-network usage.

The analytics architecture



There are four major components of the Analytics architecture:

- A scalable and reliable Hadoop MapReduce batch data processing pipeline that creates a new dataset everyday.
- Robust HBase data storage that serves data with minimal latency.
- A set of API endpoints that supports fetching data programmatically and enforces access control.
- A web application powered by an interactive UI for business users to consume data.

The raw data comes from two sources:

- Event-based data comes from Kafka and is logged onto AWS S3.
- Object-based data comes from production db.

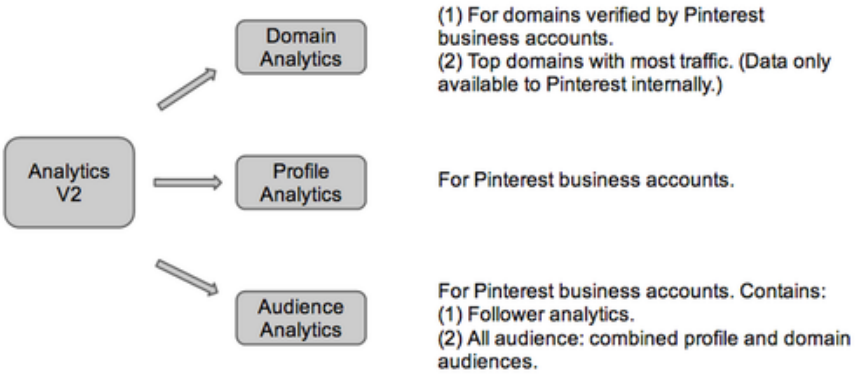
The pipeline is managed by Pinball, an internal workflow management tool.

Understanding the analytics data

Analytics provides three different datasets to the user, namely domain analytics, profile analytics and audience analytics.

- Domain analytics contains usage data about Pins or boards with Pins that link to a business domain.
- Profile analytics contains usage data about Pins or boards that belong to a business user.
- Audience analytics contains data about users or followers that interact with domain/profile content.

Pinterest analytics relies heavily on different types of user interactions, including Pin impressions, click-throughs, repins, likes and creates. We provide aggregated counts for each type of event at the Pin and board level, as well as data for Pins that are the most engaging or highest ranked in the search result with proprietary algorithms.



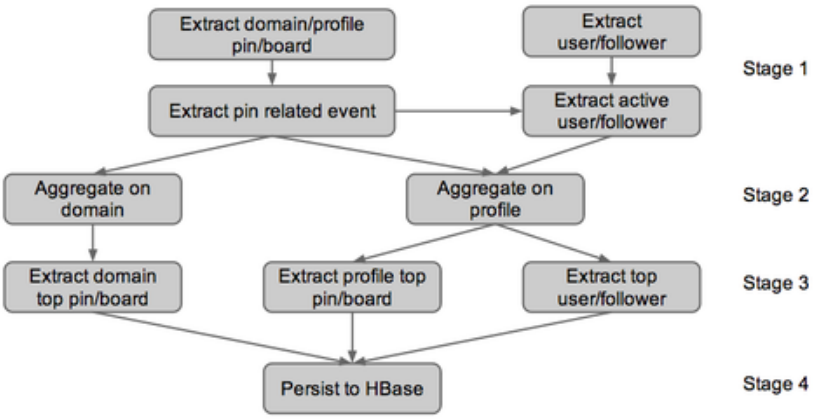
Batch data processing pipeline

To provide an accurate dataset for analytics, we built a Hadoop MapReduce data pipeline. We process tens of terabytes of data each day, so it's important to ensure the pipeline is both scalable and reliable.

The MapReduce pipeline starts to process data as soon as the data is available. It's triggered by the condition jobs which periodically check if the data is available on S3. We split up the pipeline into about 100 jobs. If some of the jobs fail unexpectedly, other independent jobs can continue processing without interruption.

There are four different stages of the pipeline, and jobs within the same stage are generally independent and can run at the same time.

Stage 1 is to extract object items, including Pins, boards and users. We schematize and only keep necessary fields since the data is heavily reused. We then extract events that are associated with those Pins, boards and users. All later stages depend on those events, so we made this phase highly paralleled and process each event type separately. In stage 2, we aggregate all events and users at the domain and profile level, and these metrics then power the daily metrics graphs. Stage 3 is for top item extraction, where we find the top items based on either event counts, or proprietary ranking criteria for each profile and domain. The last stage is to persist all data into HBase.



Tune up the processing pipeline

Since our traffic is increasing and the pipeline has a 19 hour ETA, we put in a lot of effort into making it fast and reliable. All of the data pipeline jobs are written in Hive and Cascading, and we used a few tricks to improve the performance.

1. *Optimize dependency and increase parallelism.* The general guideline for our jobs is to be as simple as possible. We store a lot of temporary intermediate results to increase data reusability. It also helps us alleviate data processing failures as we can resume the workflow at these checkpoints.

2. *Minimize data size.* Disk I/O is usually the bottleneck of processing jobs, so it's very important to minimize the data size with shared data extraction jobs, and keep only the necessary data fields.

3. *Avoid sorting the dataset.* Query clauses such as ORDER BY and DISTRIBUTED BY are all heavyweight. While processing extremely large datasets, chances are that you'll only want a limited number of top results. A better approach is to use CLUSTER BY to cluster the keys together and use a priority queue to keep only the top results.

4. *Data pruning.* For profile and domain level aggregated events and users, the dataset is usually heavily skewed, namely some profiles and domains own way more data than others. This makes it difficult for MapReduce to evenly distribute data onto reducers and hurts query performance. A workaround is to study the distribution of the dataset, and prune data entries with a small chance of appearing in the final results.

5. *Avoid or optimize joins.* Think carefully about the trade off between logging more data and spending more processing time on data joins.

When a join cannot be prevented, take advantage of map join and bucket join.

6. *Take advantage of UDF functions.* UDF functions offer great values in increasing productivity and are a scalable way of sharing query logics.

Scalable data store and serving

From the analytics data pipeline, we create a half terabyte of data each day and keep most of the data around for a month. Our storage backend is powered by HBase, which is integrated well with our Hive and Cascading processing pipeline. HBase is a key/value store, and we design our storage schema as follows:

- Row key: userparam_platform_aggregation_date
- Column family: metrics/topitems
- Column: metric names

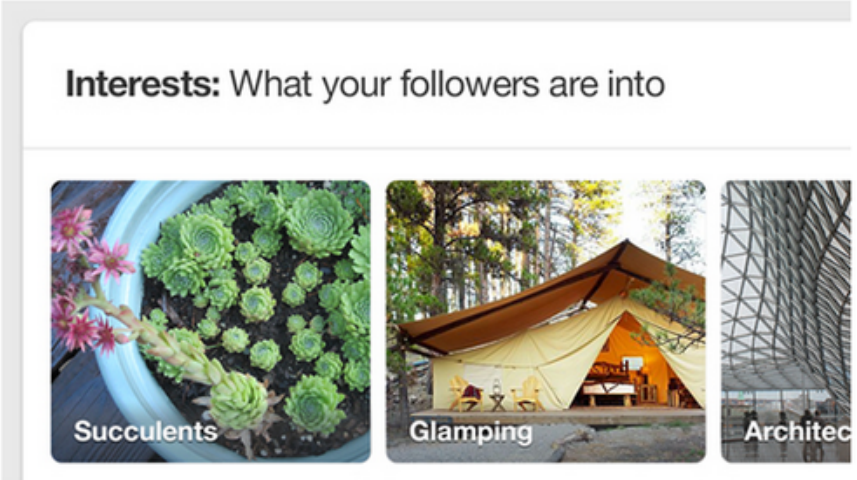
Our schema design has a few advantages, such as, the date is the last element in our key, so all of the data is consecutive when displayed in a metric graph. Also, the locality enables us to quickly fetch the data with a scan command.

We pre-aggregate data on a few levels: daily, weekly, biweekly and monthly, so we never need to aggregate any data in real time. We also pre-aggregate all available app types and store the aggregated results as separate entries. We never need to aggregate multiple userparams, so we keep it as the first element in the key, and the data will be split evenly across all region servers, and the load will be well balanced. We only have one column family in a single table because for the analytics data, it's hard to make sure that the data have similar sizes across different column families.

Surfacing data in the web application

The analytics web application is powered by API endpoints. Users with access rights can fetch the same data and do analysis on their own. The web application has rich UI components to help users dig in and discover insights.

A table of Pins with the most impressions, as well as other event types and Pin information



Analytics will now show most followed interest by audience type

You can check out the finished product now. If you're interested in working on new monetization engineering challenges like this one, join us!

Tongbo Huang is a software engineer at Pinterest.

Acknowledgements: The revamped analytics was built in collaboration with Long Cheng, Tracy Chou, Huayang Guo, Mark Cho, Raymond Xiang, Tianying Chang, Michael Ortali, Chris Danford, and Jason Costa along with the rest of the monetization team. Additionally, a number of engineers across the company provided helpful feedback.