RealtimeBoost Events - DesignDoc

last update:2017-03-08, first proposed: 2017-03-08
by: felipeg@google.com

Objective

A single place where we index and serve real world Events.

Background

Currently in production RealtimeBoostServlet runs in QRewrite detects spikes on news documents published that match the SQuery (including syns). It does so by issuing one or two RPCs to Realtime-Hivemind.

The RealtimeBoostResponse containing the Spike is sent down to Superroot in the QRewrite response and it is currently used by a few Search features (such as TopStories) to trigger faster and rank fresher documents for queries that are spiking for a given news event.

Goals

News in Search 2017 Goals

1. **Consistency** - When two similar queries have the same news intent, currently they not necessarily bring the same user experience, they might not bring the same news documents. For example, *[Oroville Dam]* and *[California Flood]* queries should lead to the same use experience during the Oroville Dam crisis. See this slide for details.

2. **Statefulness** - When a user comes back to Search 2hs after he already read the news for *[Oroville Dam]* we would like to show him only what's new about it. See this slide for details.

3. **Semantic Understanding of the Event** - When it happened, Where, Who is involved, etc...

News 2.0 - Company OKR

Trystan, our Director, is leading a new effort called News2.0. This is a complete rethinking of how google understands, index news and shows to the user.

News 2.0 will have similar concepts described above, Consistency and Statefulness is at the core of the experience. See these slides for details.

News 2.0 is one of this year's Company OKRs announced by Sundar in the Q1 TGIF.

Solution: Database of News Events

RealtimeBoost team is building a "database" of News Events by clustering different queries and spikes together. This is all updated and served in realtime.

That project + the Query Spikes will help detect the Query intent, pinpoint it to an specific Event and bring the **semantic understanding of the event**:

- Salient Terms
- Entities
- Related Queries
- Questions-Answers
- Locations
- Summary
- Labels
- Pivots
- NavBoost Queries
- Chrome Visits
- It's easy to add new dimensions such as related videos, or signals such as sentiment analysis, etc...

The Query "Oroville Dam" and the query "California Flood" will lead to the same Story right now if that's what is the Breaking News to California right now.
The next day, when another flood happened in San Jose, the query "California Flood" should lead to that story. Etc...

Statefulness - Update-me on what's new

The Events database will also make possible to have Statefulness.

We can create snapshots of these Events and show to the user only the difference between when the user came before and now. Imagine giving the user a Summary of what changed for that Story.

Events - Cluster of Spikes

We cluster two spikes together if the cosine similarity of the related Salient-Terms and related Entities set of the two reaches a certain threshold. The cosine similarity is calculated

weighted by the squashed Hivemind lift score of each token. We also require that the two Spikes overlap in time.

This approach is different from Google News Clusters, since we are grouping Query-Spike pairs based on the lift score of tokens from documents that these spikes matches. Google News Clusters uses an outdated notion of centroid unigrams of articles to cluster them purely based on the article content (centroid tokens).

Given said that, we are planning to use Google News Clusters as a signal to help building our own clusters and Google News folks (Martin Law and Maricia) are talking with us about merging the two algorithms.

This approach is already working on a demo that you can see here: go/realtime-boost-prod/firehose

Real Example:

These query-spikes were grouped together because they are talking about the same news event: *OJ Simpson is about to be release from prison in parole.*

The reason they were grouped together is because the related Salient-Terms and Entities were very similar:

Name	Score	Numerator	Denominator
ron goldman	806.35	69	191
nicole brown	745.71	63	184
brown simpson	713.45	63	196
football star	661.62	59	196
gabe grasso	631.01	47	144
oj simpson	442.90	74	504
simpson	328.67	72	738
fromong	277.70	24	52
oj	275.68	26	172
memorabilia dealers	220.47	20	51

Name	Score	Numerator	Denominator
ron goldman	1247.71	99	245
football star	1130.95	79	194
nicole brown	1104.96	81	209
brown simpson	1062.61	81	221
gabe grasso	975.94	66	178
oj simpson	839.59	110	515
simpson	620.13	105	734
early as	510.65	28	106
as early	504.01	28	108
be released	495.36	101	948

By grouping these spikes together we can build a better understanding of the news, with the queries that are spiking. We can aggregate the related Entities, Salient-Terms, and all other dimensions related to each spike.

The Clusters can track all documents about that same event and we can have "snapshots" of each Cluster to track differences on the evolution of the news (to show to the user only updates).

Code Location

Currently the code is implemented in the RealtimeBoost directory.
The two most important files are these two:

https://cs.corp.google.com/piper///depot/google3/quality/realtime/boost/ui/spike_aggregator.cc
https://cs.corp.google.com/piper///depot/google3/quality/realtime/boost/ui/common_util.cc

ModelT Serving Cluster of Spikes

After building these clusters we store in a muppet-instant using ModelT.
This ModelT will be queried by Supperroot and News 2.0 clients.

This document describes the Requirements and Scale for that Muppet instance.

This document has the implementation details for the first incarnation of our Muppet instance (ModelT).

Current state: We have a ModelT instance indexing a sub-sample of Spikes and Clusters. We are in the process of requesting capacity to increase the indexing and being able to serve to Superroot traffic.

ModelT Search

We can search/match/rank clusters on many different ways.

Refer to this doc for the full details on how the scoring was implemented.

Search by Cosine similarity of Spike

One way is to send a Spike as a query and match by cosine similarity of the Entities and Salient Terms. This is already implemented and works.
This is a powerful tool to discover the most similar Clusters based on a query that has a recent spike.

Search by Weighted set of Entities -- Or any other dimension (Location, DocIDs, Locale, Domain, etc...)

In a similar way, we can use a group of entities (with weights) and do a cosine similarity only on the entities.

Search by SQuery

We can search by regular SQuery, including receiving a big list of entities with OR.

ModelT Document Format

We store an attached proto with the aggregated version of the Cluster (not the individual spikes). We calculate it by running the liftscore again using all the Salient Terms from all the Spikes within the same Cluster. We do the same thing again for all related dimensions.
The code to calculate it is here.

The document can be indexed with the top salient terms, top queries and top entities as document tokens so the regular SQuery can match these terms. See field "content".

```
message RealtimeBoostEventDocument {
  // Stored event. Required.
  optional RealtimeBoostEvent event = 1
      [(indexing.moonshine.generic.field_spec) = {stored: true}];

  // Tokens from event.result_group().result().related_dimension_result(ENTITY),
  // event.result_group().result().related_dimension_result(SALIENT_TERMS),
  // and event.result_group().query().query().
  // This field will be tokenized using structuredsearch::HtmlTokenizer. See
  // document http://g3doc/indexing/moonshine/generic/g3doc/doc/search\
  // _api.md?cl=head#text-fields.
  // The tokens are for optional match during retrieval.
  optional string content = 2 [(indexing.moonshine.generic.field_spec) = {
    match_within_field: true
    match_globally: true
    string_options{tokenization_mode: TEXT}
```

```
}];

...
```

ModelT Serving and Google News Clusters

Our ModelT infrastructure can be used to serve even the regular Google News Clusters. If we don't merge two algorithms (RTBoost Spike Cluster with Google News Clusters), we can still use the ModelT infrastructure to Search and Serve News Clusters -- But it's unclear this is necessary given that News Cluster Id is already annotated in all documents and can be served via the normal WebMain muppet instance.

Events Timeline

The power of matching by cosine similarity of the related Salient-Terms and Entities for any given Spike can be used to bring all the Clusters related to a recent Spike/Event.

If we don't restrict by time and lower the threshold for similarity **we can bring all Clusters related to California and Flood, even the ones not related to the Oroville Dam, and thus, we can build a timeline of California Floods**.

Then client can sort it and use it anyway it wants.

Pubsub of Spikes in Realtime

In order to build these Clusters we need to discover the query-spike pairs in realtime.

Alternative Considered (demo)

The way we currently implemented the demo is by randomly sampling GWS logs and InstantNavboost queries and issuing the query to Hivemind. This has many problems. The first one is obvious, we are issuing the query to Hivemind again, when we already had done that in QRewrite. The second problem is the intrinsic delay in the GWS logs generation and sampling. We won't see the most fresh queries that just started trending.

Proposal using Pubsub from QRewrite

My proposal is to simply publish in a pubsub the RealtimeBoostResponse generated in QRewrite only for a sub-sample of queries that are trending (less than 0.5% of all queries). ONLY FOR NON LOGGED IN USERS.

RealtimeBoostServlet in QRewrite already sends an RPC to Hivemind to detect the spike. We we can simply send the RealtimeBoostResponse into a Pubsub.

Another Alternative Considered: Publish to Pubsub from Superroot

Superrot receives the QRewrite response which contains the RealtimeBoostResponse. We could implement a graph builder in PUM (Pre-Universal-Mixer), or before that, which captures the QRewrite response and publishes the RealtimeBoostResponse to a pubsub.

This option has no advantages from the above proposal, and we will have to introduce new code and implement a new graph builder in Superroot just for doing that. QRewrite already runs our code (our servlet) and seems like a natural place to publish it.

We don't currently have any code that runs for every query in Superroot (only the clients such as Stream/TopStories have it).

Privacy

If we publish only queries from non-logged in users, and only queries that are spiking, we guarantee that no PII data is published in that pubsub for 3 reasons:

1. In order for a query to be spiking, it needs to match at least 10 news documents published on 5 or more different domains (news outlets). The docs have to match the whole query (syns are allowed).
2. We limit the query word length to 5 so we limit the possibility of long-tail and unique queries.
3. Only non-logged in queries will be used.

QRewrite Pubsub Requirements and Scale

The average QPS of all queries that are trending/have a spike is about **1.5KQPS** (for whole world traffic) with short peaks of up to 5KQPS. http://shortn/_KPJzOq7Urb

The average size of a message published is **10Kb**, so total is **50MB/s** on the peak which is the max for the freebie quota from pubsub.

We can subsample that to much less if needed and only publish an small portion of the trending queries.

The call to pubsub publish in QRewrite can be async in a fire-and-forget manner, so it won't impact latency. Pubsub will also not interfere (not block) in case we are publishing too many messages per second, we can simply drop messages (or subsample).

Latency impact: None

The call to pubsub Publish in QRewrite can be async in a fire-and-forget manner, so it won't impact latency. Pubsub will also not interfere (not block) in case we are publishing too many messages per second, we can simply drop messages (or subsample), see below.

Memory impact: None

If we drop the messages when Pubsub publish is throttling us, we won't affect memory. See this pubsub doc. We can detect that the publish is being throttled by calling GetTotalPending() and we can decide to not call Publish and drop the messages.

CPU impact: Probably negligible

Pubsub publish doesn't take much CPU.

Network impact: 50MB/s added for whole world

Dividing by 15 Datacenters (where qrewrite.web runs), **3.3MB/s per datacenter**

Clusterizer Binary

We currently run a single-instance, in-memory, custom binary that listens to the pubsub of Spikes and cluster them together. We can have multiple instances sharded by locales.

The current demo instance only keeps the last 24 hours of spikes. The memory usage is about 5GB, but there are many improvements we can make to improve it. The current implementation uses a few in-memory data structures to make it extremely efficient for CPU (not necessarily efficient for memory).

There is no need to keep really old clusters in memory, since we require that the new spike time overlaps with the cluster time, and we only listen to new spikes.

We need to add a BigTable (in-memory + disk) to keep the complete data for the Clusters in case the Clusterizer dies, it needs to restart from the previous state. The BitTable, in conjunction with a master election algorithm can allow us to smoothly handle PCRs, and run the same binary in multiple cells (but one cell being the master).

This custom binary allows us to implement and test really fast.

Clusterization Algorithm

The current algorithm is greedy:

1. For a new Spike, if there is an existing Cluster that is similar and have an overlap in time with the new Spike, we add it to that Cluster.
2. If not, we create a new cluster (initially with a single spike).

Hierarchical Cluster

By using the same Cosine similarity and time-overlap function, we can easily implement a hierarchical clustering algorithm, even using some pre-existing google3 libraries such as this one.

Cluster Stability

Stability is desirable for the clients to be able to implement Statefulness and Consistent user experiences.

The current greedy implementation is by definition stable, since we never split or merge clusters.

Also the Spike timespan restriction make it unlikely that the cluster would need to split (even if we implemented it), since spikes capture a very fine granularity of events.

But even if we use Hierarchical Cluster we would like to keep the clusters stable by giving the client back the cluster always at the same level of hierarchy. But we need to think about the merge and split cases (if we ever implement it).

Long Running Cluster

The RealtimeBoost Spikes are, by nature, short lived. They have an specific time when the breaking news started and finished, usually not longer than 5 hours.

For long running events such as Superbowl, the spikes will capture the sub-events inside that big event.

We need to design an algorithm to capture all the small spikes that are related to the same big event. One idea described above is to build a timeline based on the cosine similarity between clusters/spikes that don't overlap in time. For example capture all the clusters that have Superbowl as their main entity/salient term even though they are all spread across a month (as opposed to a few hours).

I expect that Google News Clusters can help with this kind of long running clusters.
I hope Martin and Alex will be able to help here.

Whatever algorithm we devise to capture this long running cluster, we can use the ModelT data to process and re-index the clusters. For example: search for all sub-events (small clusters) and group them together into a bigger cluster. We then re-index the bigger cluster as a new kind of document in Muppet that we can query with some special restrict-token.

This way we can query Muppet to get the big-event or the sub-events for a given big-event, etc...

Single Document

When a single good document is published far away from the event that it refers to, we need to find a way to connect that document to the cluster.

Think about a NYTimes editorial analysis of some important event that happened past week…

We might try using the similarity of Doc Salient Terms to search for clusters, and/or use News Clusters.

These documents will stay in the regular Muppet and will be retrieved and ranked the same way we currently do with the FCS/Stream ranking. The Cluster association can be done in Goldmine with an Annotator or using Raffia Signals pipeline.

Design Overview