PocketTrend: Timely Identification and Delivery of Trending Search Content to Mobile Users

Gennady Pekhimenko
Carnegie Mellon University
Pittsburgh, PA
gpekhime@cs.cmu.edu

Dimitrios Lymberopoulos, Oriana Riva,
Karin Strauss, Doug Burger
Microsoft Research, Redmond, WA
{dlymper,oriana,kstrauss,dburger}@microsoft.com

ABSTRACT

Trending search topics cause unpredictable query load spikes that hurt the end-user search experience, particularly the mobile one, by introducing longer delays. To understand how trending search topics are formed and evolve over time, we analyze 21 million queries submitted during periods where popular events caused search query volume spikes. Based on our findings, we design and evaluate *PocketTrend*, a system that automatically detects trending topics in real time, identifies the search content associated to the topics, and then intelligently pushes this content to users in a timely manner. In that way, PocketTrend enables a client-side search engine that can instantly answer user queries related to trending events, while at the same time reducing the impact of these trends on the datacenter workload. Our results, using real mobile search logs, show that in the presence of a trending event, up to 13–17% of the overall search traffic can be eliminated from the datacenter, with as many as 19% of all users benefiting from PocketTrend.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Filtering

Keywords

Web search; trend detection

1. INTRODUCTION

Search portals like Google and Yahoo are among the top daily visited websites, indicating how crucial these services are to end users. In response to their popularity, search engine providers have put tremendous effort in optimizing their infrastructure, usually either through server-side [2, 29, 4, 7, 18] or client-side caching [15], to enable the fastest possible search user experience.

Unfortunately, these optimizations can be rendered ineffective or costly due to the unpredictable nature of the search query volume. Important events that take place in the physical world can instantly translate into significant, and often unpredictable, spikes in search requests. Figure 1 shows Bing's normalized hourly mobile query volume during the week of the 2012 presidential elections in the United States. The blue line represents the actual query volume, while the red dotted line shows the estimated query volume when all queries related to the elections were manually filtered out. On the election day, the search volume increased by roughly 30% compared to the previous days, and remained that high for several hours. When all queries related to elections are filtered out, the search volume falls back to the same volume levels as the previous days, indicating that the volume spike was due to the elections.

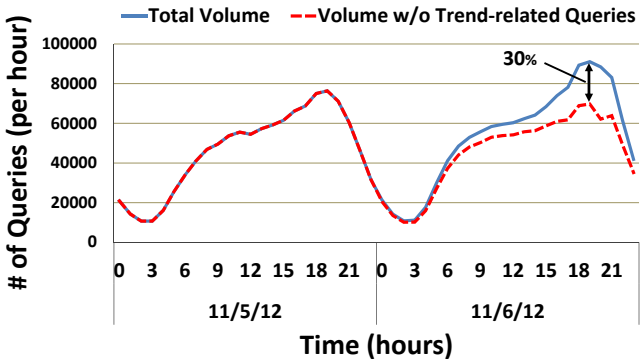


Figure 1: Normalized hourly query volume during the week of U.S. president elections (11/6/2012) with (blue line) and without (red dotted line) the election-related queries.

Even though not happening daily, search volume spikes as in Figure 1 have significant impact on the end user experience. First, in the presence of such unpredictable events, end users lose the benefit of real-time search experience delivered by previously proposed approaches for client-side search index caching [15] that is orders of magnitude faster than the actual service call. These approaches have focused on periodically (e.g., daily) caching the static part of the search index, and therefore cannot handle fresh content that might spike during the day. The impact on the end user experience is particularly crucial to mobile devices, where cellular links are limited by lengthy connection setup times [8].

Second, even though server-side caching techniques [19, 29, 4, 7, 3] can help smoothen out the datacenter's computation overhead due to these search volume spikes, they do nothing about reducing the number of search requests reaching the search backend. As the number of requests simultaneously hitting the backend increases, the datacenter load gets closer to its capacity limits. This usually causes a slower user experience, makes the search engine more susceptible to common Denial of Service (DoS) attacks, and in the worst case can bring the whole service down incurring a large revenue loss. To overcome these risks, search engines invest heavily on over-provisioning their datacenters' capacity by adding more servers than they actually need on a daily basis which leads to significant increase in datacenters' cost.

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741641 .

In this work, we study the feasibility of enabling a real-time search experience for trending search topics without overwhelming the search backend with an excessive number of search requests. We envision search engines that can timely detect and efficiently propagate trending search content (i.e., search queries and corresponding search results) to users' mobile devices to enable a real-time search experience at a lower cost for the datacenter. With such a mechanism in place, in the case of the 2012 U.S. presidential elections (Figure 1), 30% of users' queries could be instantly served locally (e.g., through the web browser or a dedicated search application), without sending a request to the search engine. Likewise, 30% of the overall query volume could be eliminated from the search backend at peak time.

To enable this type of user experience, the search backend must address three challenges. First, it needs to detect, in real-time, search query volume spikes due to trending events. Second, it must identify the part of the search index associated to these events. Third, it must proactively and efficiently propagate the identified trending search content to end users in a timely manner. Note that the proactive dissemination of trending content to end users can introduce additional request and/or bandwidth overheads if not done properly. For instance, proactively pushing trending content to every single user of the search engine is not an option as it would cause the number of connections and bandwidth consumed by the datacenter to explode. Instead, the search backend should be able to intelligently decide *which* users and *when* should be updated with the appropriate content to avoid increasing the number of search requests and to keep the bandwidth overhead as low as possible.

To address these challenges, we conduct a data-driven design exploration study. To understand how trending search topics are formed and how they evolve over time, we analyze 21 million mobile search queries submitted to Bing during periods where real events led to search query volume spikes. Using this dataset, we quantify the benefit and bandwidth overhead of different approaches to proactively pushing trending search content to end users. Based on our findings, we design *PocketTrend*, an architecture for enabling real-time search experience for trending topics. Its design is based on key observations of the search log analysis:

- Only a small number of search queries and search results is responsible for the majority of a trending event's query volume. PocketTrend leverages this observation to employ a two-step algorithm for identifying the small part of the search index that corresponds to the trending event, reducing the footprint of the data to be pushed to users to roughly 1 MB.

- Many users access the search engine with queries unrelated to the trending event, *after* the trending event has been detected. PocketTrend leverages this finding to opportunistically update these users by piggybacking the trending search content along with the requested search results. In that way, the datacenter does not need to establish any new connections to deliver the trending content. Using this approach, we show that up to 13-17% of the overall search traffic (roughly 50% of the trending search traffic) can be answered locally on the users' devices with a reasonable bandwidth overhead.

- Frequent users that accessed the search engine right before the trending event started are likely to search for the trending event in the future. However, we show that the overhead of pushing trend content to these users in terms of number of connections that the datacenter must maintain and bandwidth it must consume makes this approach infeasible.

Event	Log dates	#users	#queries
U.S.Pres.Elections, 11/6/12	2–9 Nov.12	700K	7.7M
PopeElection, 3/13/13	13–17 Mar.13	550K	7.0M
BostonMarathonBomb, 4/15/13	14–21 Apr.13	660K	6.7M

Table 1: Details of the mobile search dataset used to study users' search behavior around trending topics.

2. SEARCH LOG ANALYSIS

Timely detecting and propagating trending search content to end users poses three fundamental challenges: *What* content to push, *When* to push it, and *Whom* to push it. Addressing these challenges requires first understanding how trending topics are formed and evolve over time. To gain insight on how mobile users search for trending events, we analyzed more than 21 million search queries, submitted by roughly 2 million unique users in the United States. All queries were submitted through mobile devices to Bing. In particular, we leverage three week-long datasets, each corresponding to a significant physical event that resulted into a major trending topic as shown in Table 1. The events are U.S. president elections, pope election, and Boston marathon bombing.

Each entry in the search logs has 6 fields: user ID, search query submitted by the user, timestamp, list of search results that were clicked by the user, and all search results that were shown, but were not clicked by the user. A search result consists of a URL and a snippet describing the URL.

To prevent disclosing sensitive information about Bing's exact query volume, at given times for the commercial search engine that collected these logs, the query volumes have been properly normalized. This normalization does not affect the results presented throughout the paper as we look at relative gains.

2.1 What to Push

First, it is necessary to automatically detect trending events near real time, and also identify the portion of the search index that is related to these events. This portion of the search index will become the actual search content (search queries and corresponding search results) that will be pushed to end users. The feasibility of this approach depends on how concentrated the search content associated to a trending topic is. If users' clicks are evenly distributed over a large number of search results, then identifying the trending part of the search index can be quite difficult. Conversely, if most users click on a small number of search results when searching for trending topics, then the trending search content is more compacted and therefore ideal for client-side caching.

Figure 2 shows the cumulative coverage achieved by the most popular 50 search results for the three trending events we studied. To generate these data, we extracted all queries related to each trending event, and then identified the most popular (most often clicked) search results. We find that the single most popular search result accounts for anywhere between 4% and 7% of the overall trend-related search result clicks depending on the trending event. All together, the 50 most popular search results cover anywhere between 24% and 30% of the overall trend-related search results that were clicked by users. In other words, simply pushing the top 50 search results associated to a trending topic to end users, could result in 30% of users' queries related to the trending event being answered instantly on their mobile devices without the need to reach the search engine.

When considering domains instead of absolute URLs (e.g., cnn.com/story1.html and cnn.com/story2.html are mapped to the same domain cnn.com), the entropy of the search results is even smaller. The top 50 domains account for 52%–59% of the over-

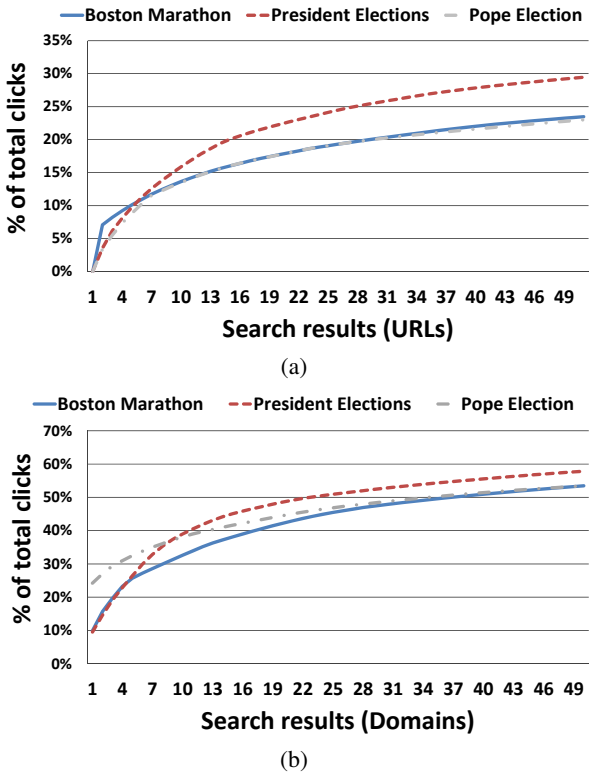


Figure 2: The cumulative percentage of the overall trending search results for the 50 most often clicked trending search results when considering: (a) absolute URLs, and (b) URL domains.

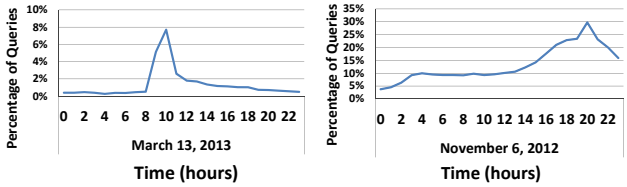
all query volume depending on the event, suggesting that caching most popular URLs can be efficient.

Summarizing, Figure 2 shows that a small number of search results (URLs) accounts for the majority of the trending search volume, indicating the feasibility of the proposed approach. Section 3.2 describes how PocketTrend exploits this trend to identify a small set of trending search queries and search results to push to users. The larger the size of the content pushed to the users, the more likely the case of satisfying users' search requests locally (on the device) in real time. We evaluate this tradeoff in Section 4.5.

2.2 When to Push

Assuming the set of search queries and corresponding search results is identified, a decision about when to push this content to end users needs to be made. Even though a trending event can be detected quite early (within 20 minutes), the content related to the event might evolve over time, especially in the very beginning. Hence, pushing the trending data too early could be ineffective as crucial content might be missing.

Figure 3 sheds light on the distribution of all trend-related queries over time for pope and U.S. president elections. In particular, this figure shows the percentage of the total hourly query volume that corresponds to trend-related search queries over the lifetime of the event. For both events the peaks have durations that span several hours. However, it is clear that different trends evolve in different manners. During the pope election event, there is an immediate sharp 3-hour window where the event is mostly active, and then it stabilizes and slowly decays over time. On the other hand, the U.S. president election event starts with a moderate spike and



(a) Pope election (b) U.S. president elections

Figure 3: Percentage of the total hourly query volume that is related to the trending event over time.

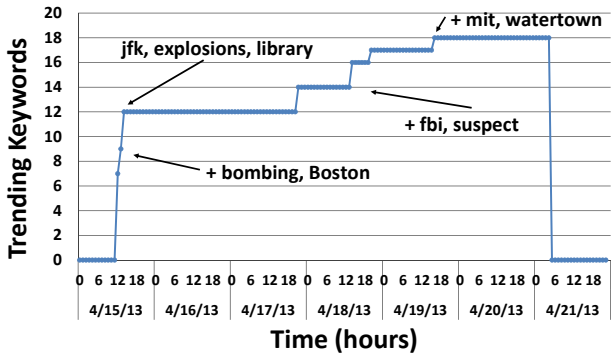


Figure 4: Evolution of trends' keywords over time for the Boston Marathon bombing event.

remains stable for multiple hours until it reaches its peak that lasts roughly 8 hours. We also observe that most trend-related queries are submitted during each event's peak that lasts a few to several hours. Hence, the window of opportunity for pushing trend content to users is only a few hours.

Finally, the trending content can evolve during the trending event lifetime. Figure 4 shows the number of keywords over time for the Boston Marathon bombing event; in the first 2 hours, 7 keywords have been detected, but, by the time the event has stabilized, 18 keywords have been detected.

Summarizing, to address the constant evolution of trending events, PocketTrend needs to continuously scan the search logs to identify the freshest trending search content and push it to users. In Section 3, we propose various techniques to achieve this goal.

2.3 Whom to Push to

Identifying which users, and in what order, will be updated with the detected trending search content is crucial. This is not a trivial task as commercial search engines have hundreds of millions of unique users. Attempting to simultaneously update every user with the trending search content will overwhelm the data center, creating a bigger problem than the one PocketTrend solves.

Mobile users can be updated "actively" or "passively". Active updates require the datacenter to track each user and preemptively push the trending search content when it becomes available. Such an approach, however, can lead to pushing content to large numbers of users out of which only few might eventually search for it.

Given that, if not highly accurate, active pushing can further stress the datacenter's I/O resources, we also explore passive, opportunistic updates. Each time a user submits a query to the search engine and a trending search topic is currently active, the search engine can opportunistically update the user with the latest trending search content along with providing the search results to the

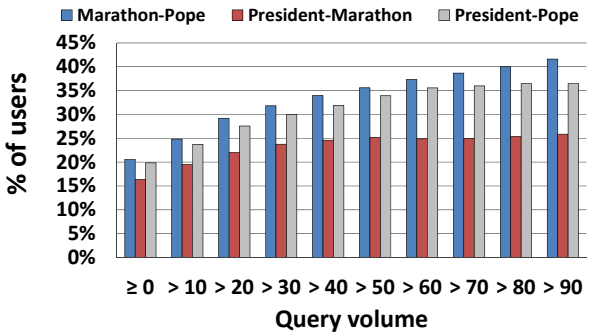


Figure 5: Percentage of users that searched for both trending events for every pair of the three trending events we studied. Users are categorized by query volume.

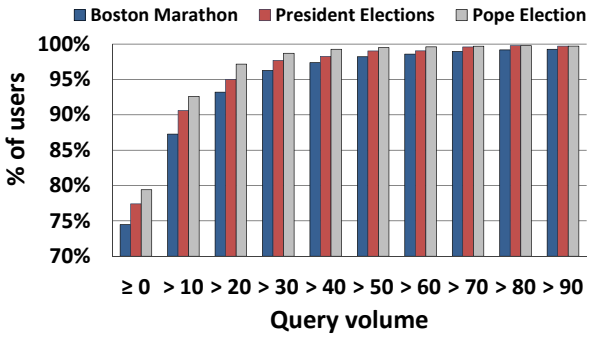


Figure 6: Percentage of users that submitted a search request within 2 hours before the trending event happened, and then searched for the event itself after it happened.

(non-trend-related) query. In that way, no extra requests need to be submitted to the datacenter and no user-tracking is required.

To understand who should be updated with the trending search content and how, we leveraged the search logs to explore the feasibility of both active and passive approaches.

Active Content Pushing. First, we opted to identify loyal users of the search engine since loyal users are more likely to come back and search for trending topics. We defined loyal users as users that had submitted a large number of queries in the past and they had also searched for at least one trending event. The intuition is that if a user searched for a trending event in the past, then most probably that user will also search for the next trending event. Figure 5 shows the percentage of users that searched for both trending events as a function of user query volume, for every pair of the three trending events we study. Note that as the user query volume increases, the percentage of users that searched for both trending events becomes higher (indicating a stronger correlation). However, the user intersection between trending events is rather low; about 25% of the users that searched for a previous trending event will search for a future one. This could be because some users might search for trending content on multiple devices, may not be interested in all trending topics, or may have already read about the event in the news. As a result, updating users that previously searched for a trending event would result into lots of user updates to the benefit of only a small fraction (≈25%) of users.

The second approach to identifying whom to push to the trending search content was based on the assumption that users that recently submitted a search query in the search engine are likely to search for the trending content as soon as they find out. Figure 6 shows the percentage of users that submitted a search query within 2 hours before the trending event started, and they also searched for the trending event after the event took place. Again, the higher the query volume of a user, the higher the probability that the user will eventually search for the trending topic. However, in this case, more than 80% of the users submitting a query within the last 2 hours will return to search for the trending topic.

Summarizing, high volume recent users are ideal candidates for pushing the trending search content. The number of users a search engine can push content to ultimately depends on the datacenter's bandwidth available for PocketTrend updates. We analyze this tradeoff in Section 4.2.

Passive Content Pushing. Table 2 shows the percentage of mobile users that submitted an unrelated query after the trending event happened, and then returned to search for the trending topic. For these users, the unrelated query can be used to piggyback trending search content. Surprisingly, up to 26% of all the users that searched for something while the trending event was active later search for the trending event as well. This corresponds to 79% of the users that eventually searched for the trending event. This correlation demonstrates the opportunity to exploit simple and low-cost passive updates to opportunistically push trending content to users.

Event	% of returning users out of	
	all users	users that searched the trend
U.S. president elections	18.2%	77.4%
Pope election	3.1%	79.4%
Boston marathon bombing	26.3%	74.5%

Table 2: Percentage of users that searched for an unrelated topic when the trending event was ongoing and then came back to search for the actual trending event.

Summarizing, passive updates could complement or even replace active updates to enable a real-time search user experience with a lower impact on the datacenter's bandwidth requirements. In Section 4.3 we analyze this tradeoff.

3. POCKETTREND ARCHITECTURE

The design of the PocketTrend architecture is driven by the findings of the search log analysis described in the previous section. As shown in Figure 7, PocketTrend runs on the search backend and consists of three major components: *trend event detection*, *trending content identification*, and *trending content delivery* to end users.

First, like in previous work [20, 24, 9], PocketTrend detects trending keywords by analyzing the frequency at which these keywords are used in the search queries. The goal is to detect keywords that are searched significantly more frequently than normal (e.g., five times more frequently than during the same hour the day before). The detected trending keywords are grouped together to form trending events. A trending event at this stage is nothing more than a collection of frequently-searched keywords.

At the next stage (trending content identification in Figure 7), PocketTrend scans the search logs to identify all queries that contain a significant number of trending keywords, and marks these queries as trending. A subset of these, the ones that have the highest search rates and the highest number of clicks, is selected to form the trending search content to be pushed to users. Likewise, the most clicked search results for the selected trending queries are also added to the trending content. This strategy ensures that only the most relevant trending search content will be propagated to users' devices, saving bandwidth and storage resources.

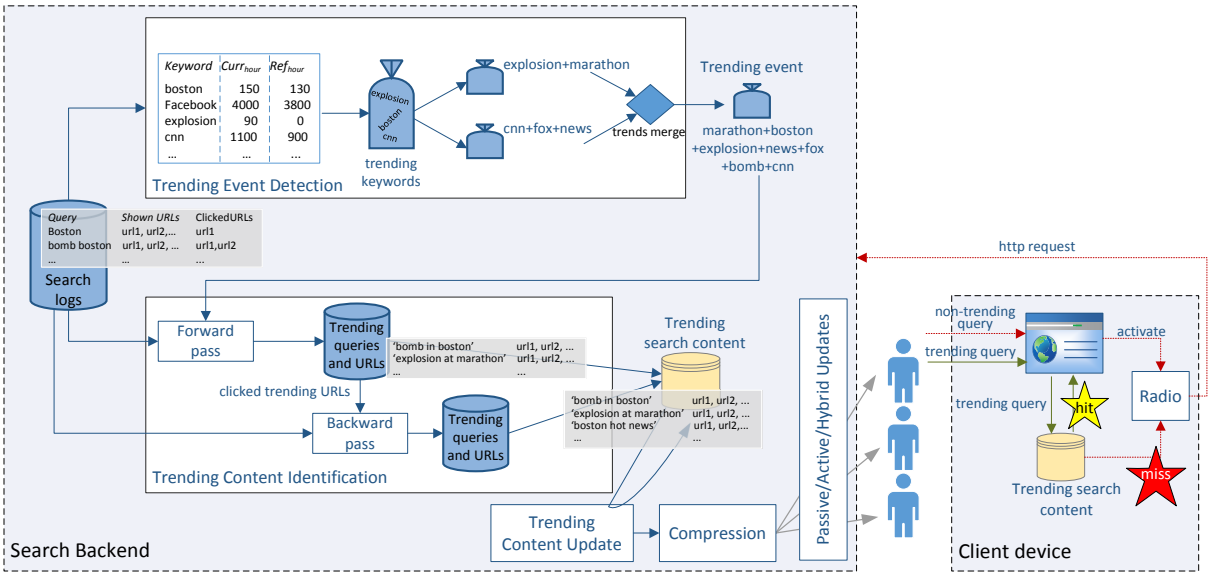


Figure 7: PocketTrend overview.

Trend event detection and trend content identification continuously run in the search backend. In the last stage, the freshest trending search content is pushed to end users actively or passively (see Section 2). To reduce the bandwidth consumed for pushing the trending search content, PocketTrend also uses data compression techniques [31].

At the client side, the trending search content can be used to enable real-time query suggestion or query auto-completion. Since both queries and search results are pushed to the client devices, query suggestions or auto-completions can be accompanied with the relevant search results that the user can instantly click on. If these are not satisfactory the user can still select a query or manually enter his own to get the freshest search results from the search backend. In that way, PocketTrend does not replace the search engine, but it rather works along with it to enable the fastest possible search experience for end users.

In the following, we describe all PocketTrend components in detail.

3.1 Trending Event Detection

Trending events are identified in two steps. First, an initial set of trending keywords is identified, and then trending keywords are grouped together to form events.

3.1.1 Trending Keyword Detection

As in related work on trend detection [20, 24, 9], we leverage the observation that in the absence of active global trends, the same keyword within the same hour (but for different days) tends to have similar numbers of appearances in search queries. We define a keyword to be trending if the number of appearances within a specific hour in the current day is significantly higher (e.g., 5x more) than in the same hour during a reference day.[1] Formally, a keyword is defined to be frequent if

$$\frac{Curr_{hour}}{Ref_{hour}} \geq KeywordRatioThreshold \quad (1)$$

where $Curr_{hour}$ is the number of appearances of a keyword in the current hour and Ref_{hour} is that of the same keyword in the same hour in a reference day.[2]

$KeywordRatioThreshold$ in formula 1 is an empirically defined threshold that separates random, small frequency variations of keywords that can happen across days, from significant frequency variations that indicate a trending event. We are interested in major trending events that result into significant search volume spikes as in Figure 1, so finding an appropriate value for $KeywordRatioThreshold$ becomes trivial. Based on our analysis of real user search logs, we found that keywords that appear more than 5 times more frequently than usual indicate the presence of strong trending events causing search volume spikes.

Even though Ref_{hour} in formula 1 has a non-zero value for most keywords, there are always keywords that appear suddenly due to a trending event. For instance, keywords like “pope" during Pope election, and “explosion" during the Boston marathon bombing were completely new keywords ($Ref_{hour} = 0$). To handle such cases, and to differentiate actual trending keywords from random new keywords, a keyword with $Ref_{hour} = 0$ becomes trending only if it is statistically important, in the sense that it is observed at least in 0.1% of all queries in the last hour.

3.1.2 Trending Keywords Grouping

Initially, every trending keyword is a trending event by itself. However, the detected trending keywords could belong to one or more trending events taking place at the same time. To properly group keywords together to form trending events, we examine how frequently trending keywords are searched together. Intuitively, the more often two keywords are searched together, the higher the probability they refer to the same topic. For example, for the Boston marathon bombing event we observed that the words “marathon" and “explosion" were searched together 93% of the time. Empirically, based on the analysis of millions of mobile search queries, we found that trending keywords that are searched

[1] We use a previous day as reference point to identify trending keywords. We also tested time periods smaller than an hour and a “sliding window” approach, and achieved identical performance.

[2] There are other possible ways to define the word frequency, e.g., based on tf-idf metric [12, 25], but we found that those techniques usually incur higher computation overheads while generating same/similar list of keywords.

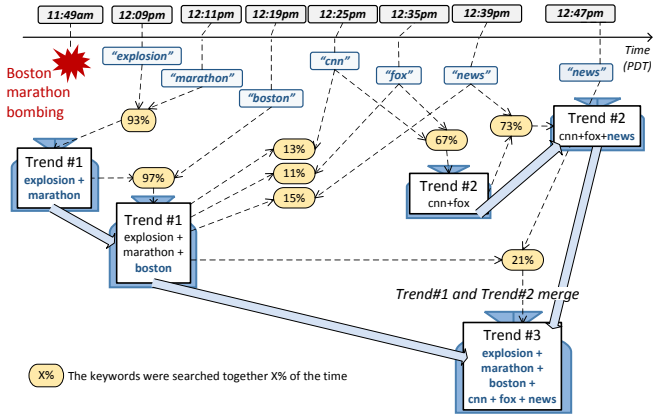


Figure 8: Detailed example of trend detection and trending keywords grouping (Boston marathon bombing).

together more than 20% of the overall times they appear in the search logs, they refer to the same trending event. As a result, every pair of trending keywords that is searched together at least 20% of the times they appear in the logs is merged to form a single trending event. Every new trending keyword is also evaluated against all the current trending keywords, and it either joins an active trending event or forms a new one.

The detected trending events are not considered as immediately active. They become active when the overall percentage of queries in the last hour that contains at least one of the trending event's keywords is higher than 1% of the overall search traffic. We impose this threshold to ensure that content is pushed to users only for events that are large enough to matter (for the user and for the datacenter).[3] The threshold could be adjusted depending on the datacenter's requirements.

To illustrate how trend detection works in practice, Figure 8 shows the trend evolution for the Boston marathon bombing event. The explosion happened at 11:49am PDT. Within 20 minutes, the keyword "explosion" is detected as trending, followed by "marathon" in 2 more minutes. This creates the first multi-keyword trend as these two keywords are searched together more than 90% of the time. In 8 more minutes, the keyword "boston" is detected as trending, and it is added to the initial trend. Later on, the keywords "cnn", "fox" and "news" are also detected as trending, but these keywords are not added to the existing trend #1, because they are not searched frequently enough together with the keywords from the current trend (13%, 11% and 15% correspondingly, at the time of detection as trending). At the same time, these trending keywords are frequently searched together, which allows us to create a new trend (trend #2). In 8 more minutes (at 12:47pm) the trending keyword "news" passes the threshold for intersection with trend #1, which leads to the same keyword being in two active trends. This leads to merging both trends into a single trend that is later used for identifying the trending search content.

3.2 Trending Content Identification

At this point, a trending event is a collection of trending keywords that are frequently searched together. PocketTrend leverages this collection of keywords to identify the search content related to the trending event. By search content, we mean all the search queries related to the trending event, and the search results associated to these queries. Detecting this content is quite challenging for two reasons. First, mobile users search for a specific topic in multiple different ways due to either the large number of synonyms used, or the typos and grammatical errors (e.g., "boeton explo"). Second, queries that look very similar can semantically be very different. For instance, the queries: "boston marathon", and "boston marathon bombing" could be very different as the former is an informational query about the event, while the latter focuses on the bombing incident.

3.2.1 Detecting Relevant Queries: Forward and Backward Passes

We address this problem with a two-step process, shown in Figure 9. First, a *forward pass* of the search logs takes place to identify a small set of queries that with high certainty are related to the trending event. Given this small set of highly relevant queries and the search results that users click on in response, PocketTrend identifies the core set of search results related to the trending event. At the second step, a *backwards pass* of the search logs takes place where we identify additional queries related to the trending event based on the set of search results identified in the first step. In particular, all queries that resulted into users clicking on a search result related to the trending event, are automatically assumed to be related to the trending event. This way, even infrequent and unconventional queries used to search for the trending event (e.g., "boston explode") can be captured.

Given that the goal of the *forward pass* step is to identify a small number of search queries that are relevant to the trending event with high probability, PocketTrend follows a conservative approach. All search queries in the past hour that contain 3 or more trending keywords are automatically assumed to be related to the trending event. Since mobile users' queries contain 3-4 or less keywords most of the time [13, 14], 3 trending keywords indicate a strong match.[4] All search queries that contain zero or just one trending keyword are ignored in this step, because even generic keywords such as "Boston", "cnn" or "news" can become trending. Search queries that contain exactly two trending keywords, the most common case in the mobile search logs, can be more complicated. For example, the queries: *"boston marathon results"* and *"explosion at boston today"* contain exactly two trending keywords for the Boston marathon bombing event. However, the first query is most probably not related to the bombing, because the user cares about the marathon results, and not the bombing incident.

To handle these cases, PocketTrend leverages the intuition behind techniques like tf-idf [25, 23], used for measuring how important a word is in a document. At a high level, the tf-idf value of an input text increases proportionally to the number of times the words composing it appear in a reference dictionary (term frequency), but is offset by the frequency of the words in the dictionary (inverse document frequency), which helps to control the fact that some words are generally more common than others. To achieve the same effect of inverse document frequency, we implement a matching algorithm where different trending keywords are given different weights in the matching process depending on how *unique* they are. For example, the keyword "boston" can be used in many different queries (e.g., "boston weather"), and hence its uniqueness is lower than the keyword "explosion". We can distinguish these two cases through the keyword frequency increase between the trending event's time period and a reference time period

[3] We admit that our methodology can potentially miss some trending events that do not exhibit significant increase in the search traffic, but we expect the effect of such relatively small trends on the datacenters to be minimal.

[4] In a more general case, the number of words for a strong match can be defined based on the relative number of words in the trending event.

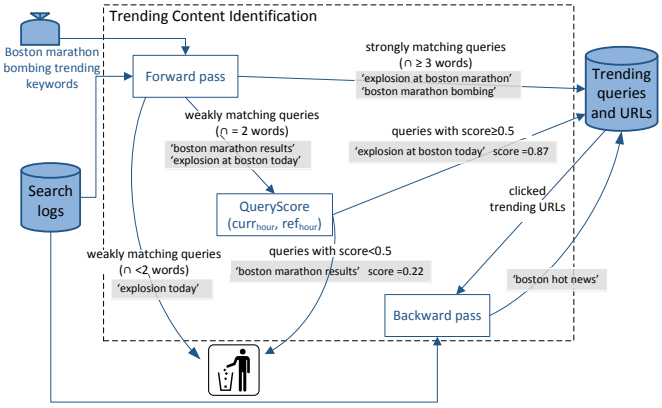


Figure 9: Trending content identification using forward and backward passes on top of search logs.

(formula 1). Keywords like "boston" usually have low increase ratios ($5x$–$8x$), while keywords like "explosion" can have ratios over $100x$.

Given this observation, PocketTrend computes a matching score $Query_{score}$ for every candidate search query:

$$Query_{score} = \frac{\sum_{Keyword-k_i InQuery} Uniqueness_{k_i} * Curr_{hour}^{k_i}}{\sum_{Keyword-k_j} Uniqueness_{k_j} * Curr_{hour}^{k_j}} \quad (2)$$

where $Uniqueness_k$ represents how unique keyword k is:

$$Uniqueness_k = \begin{cases} 1, & \text{if } 5 \leq \frac{Curr_{hour}^k}{Ref_{hour}^k} \leq 20 \\ 2, & \text{if } 20 < \frac{Curr_{hour}^k}{Ref_{hour}^k} \leq 50 \\ 3, & \text{if } 50 < \frac{Curr_{hour}^k}{Ref_{hour}^k} \leq 100 \\ 5, & \text{if } 100 < \frac{Curr_{hour}^k}{Ref_{hour}^k} \leq 1000 \\ 10, & \text{if } 1000 < \frac{Curr_{hour}^k}{Ref_{hour}^k} \end{cases} \quad (3)$$

$Curr_{hour}^k$ and Ref_{hour}^k represent the number of queries in the past hour and in the reference hour respectively, that contained the trending keyword k. The higher the change in the appearance frequency of the keyword k in the search logs, the higher its uniqueness value. Note that the weight of a trending keyword (equation 2) takes into account both the uniqueness and absolute frequency of a keyword by multiplying them. For every query we find its matching weight as the total sum of the weights of all trending keywords that are contained in the query, normalized by the total weight of all available trending keywords. In that way, the matching score for a query is always a value between 0 and 1. Every query with a matching score above 0.5 is assumed to match the trending event.

At the *backward pass step*, we record all the search results that users clicked on after submitting queries that have been identified as trending. PocketTrend uses these results to identify additional queries that belong to the trending event. In particular, every query that resulted into a click on one of the search results identified in the *forward pass* step is added to the trending event.

3.2.2 *Trending Content Cache Formation*

After the trending event has been detected, a set of search queries associated to the event has been identified. By examining all the search results that users clicked after submitting these queries, the final set of search results (URLs and snippets) associated to the trending event can also be generated. When considering every single query and search result associated to the event, the total number of queries and search results can be quite large. For instance, in the case of U.S. presidential elections, the identified trending content includes 27,000 queries and 37,000 search results (Table 3). Pushing all this data to mobile users is not desirable due to the bandwidth requirements. However, as demonstrated in Section 2.1 (Figure 2), the majority of the search traffic related to the trending event is concentrated around a small set of these queries and corresponding search results (e.g., top-50 search results). As a result, PocketTrend leverages the most popular trending search queries (e.g., 1000 top queries) along with the corresponding search results to form the trending cache that is pushed to users. As we will show in Section 4.5, this approach ensures high, close to ideal, performance, while minimizing the bandwidth consumed.

Event	Queries	Results
U.S. Presidential Elections	27K	37K
Boston Marathon Bombing	13K	16K
Pope Election	1K	3K

Table 3: Number of unique trending search queries and search results for each trending event.

3.3 User Update Strategies

After the trending search content has been identified, PocketTrend needs to timely and efficiently propagate this content to users. PocketTrend explores two techniques for achieving this, based on the search log analysis described in Section 2.3.

First, it actively identifies a subset of the search engine's users that with high probability will search for the trending topic in the future, and pro-actively pushes the trending search content to these users. The selection of users is based on the analysis of the mobile search logs in Section 2.3 (Figure 6). PocketTrend identifies all users that submitted a query within two hours before the trending event took place, and pushes the trending search content to these users in descending order of their query volume. To limit the effect of active updates on the search backend and its bandwidth consumption, we enforce a maximum number of users to be updated every minute. In that way, PocketTrend does not overwhelm the datacenter by simultaneously pushing content to all users.

Second, based on the findings of the mobile search log analysis in Section 2.3, PocketTrend complements active content pushing with passive user updates. As Table 2 shows, a large percentage of users (up to 26%) submits queries unrelated to the trending event after the trending event has been detected. PocketTrend leverages this to opportunistically push the trending search content to these users along with the search results for the unrelated search query. In that way, PocketTrend can timely update users that might have never been updated through active updates, without increasing the datacenter's workload.

In the next section, we quantify the benefits and overhead for both active and passive updates, and study the feasibility of each technique in detail.

4. EVALUATION

In this section, we quantify the improvements PocketTrend can achieve and analyze the overhead paid by both the client and the server side. In particular, we leverage the search log traces from the 3 representative trending events described in detail in Section 2. On top of these 3 trending events, we also separately test a month

Trending event	% of unique users			
	PT-5k	PT-UpdatesOnly	PT-Unlimited	PT-Ideal
U.S.Pres.Elections	8.8%	10.7%	11.8%	13.7%
PopeElection	2.8%	2.3%	2.9%	4.5%
BostonMar.Bomb	19.6%	18.7%	21.5%	21.8%

Table 4: Percentage of users for which a trend cache hit is registered, when PocketTrend with active updates (5k), passive updates (UpdatesOnly), unlimited cache size (PT-Unlimited) or ideal cache (Ideal) is used.

of search logs data (from June 15th to July 15th 2013) and detected five small trends: Lil Wayne Hospitalization, Father's Day, Gandolfini's Death, USA Independence Day, and San Francisco Plane Crash).[5] By replaying these search logs, we know exactly which user searched for what, at what time, and what search result he/she clicked in response. At the same time, while replaying back the search logs, we run PocketTrend to identify trending search content and push it to end users. This enables us to know if the search query and search result clicked by a user when searching for the trending topic has already been pushed to the user's mobile device. If true, a cache hit is recorded meaning that this query did not have to reach the datacenter, and the user had an instant user experience. If either the query submitted by the user or the search result the user clicked on was not included in the pushed trending content, then a cache miss is recorded, and the query is assumed to hit the datacenter like any other search query. Note that in doing this we take into account the freshness of the search content. In fact, a user may have received trending search content but that may be too old to include the search result the user clicked on when doing the actual search, so a case like this would be recorded as a cache miss. Using this setup we evaluate PocketTrend when active or passive updates are used to push trending search content to end users.

4.1 Notation and Experiment Parameters

In our evaluation, we consider an active update mechanism (*PT-Xk*) which updates at most X thousand users every minute (e.g., *PT-5k* updates up to 5k users a minute), and a passive update mechanism (*PT-UpdatesOnly*) that opportunistically piggybacks the trending content on other pending queries' results (Section 3.3). Unless otherwise noted, in both cases every single detected trending query and search result is pushed to users (Table 3). We also compare PocketTrend's performance to a baseline system (*No-Caching*) with no client-side caching support, and to an ideal system (*PT-IdealCache*) which assumes that all users are updated with the freshest trending search content, irrespectively of its size, right before they submit their trend-related query. Note that such a system is not feasible and is only used for comparison purposes.

4.2 End user Search Experience

To understand the benefits PocketTrend can bring to users, we measured the fraction of users that received the trending search content, and subsequently searched for the trending topic by submitting a search query and clicking on a search result that was already pushed on their devices. Table 4 shows the percentage of all search engine users during the lifetime of the event that benefited at least once from PocketTrend in the case of active (*PT-5k*) and passive (*PT-UpdatesOnly*) updates. In both cases, we assume that the 1000 most popular trending search queries are pushed to the users. Results when all trending search content is pushed (*PT-Unlimited*), and when an ideal cache is used (*PT-IdealCache*) are also shown.

[5]We did this experiment to show that PocketTrend works for new testing data that was not used to define its design.

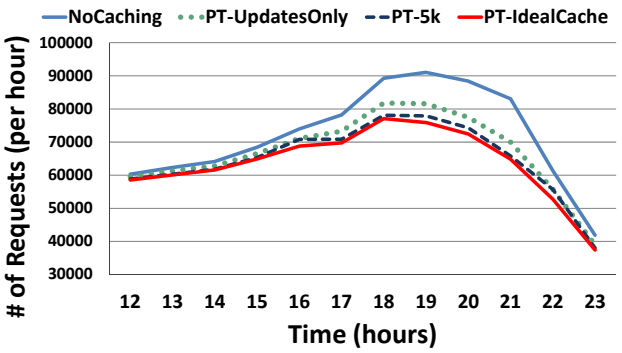


Figure 10: Search query volume reduction with PocketTrend (PT) for different update strategies (U.S. president elections).

For a large event like the Boston marathon bombing, even 20% of all unique users the search engine sees during the event lifetime benefit from PocketTrend. Given that search engines serve hundreds of millions of users on a daily basis, this percentage corresponds to several tens of million of mobile users. For smaller events like the pope election this percentage reduces to 3% of all unique users, but still corresponds to several million users. Surprisingly, passive updates only lightly reduce the effectiveness of PocketTrend, with anywhere between 2% and 19% of unique users being benefited. In the case of the U.S. president elections, passive updates actually outperform active updates. Due to the slow evolution of the trending event, active updates can push trending search content to users early during the trend development process. As the event evolves over time, updated users might search for the trending event but for content that was not captured in the active push (e.g., Ohio state election results were not rolling in when the content push took place). On the other hand, passive updates take place over time, as users come online, allowing them to use the freshest version of the trending search content, and therefore benefit more end users. This unexpectedly good performance of passive updates is crucial to make PocketTrend a low-cost mechanism for datacenters (more on this later).

Not only does the passive updates mechanism achieve similar performance to the active updates mechanism, but its performance is also very close to that of an ideal, but not practically feasible, implementation of the cache (freshest content is always available irrespectively of its size). As Table 4 shows, the ideal cache can only provide real-time search experience for an extra 2-3% of users.

4.3 Datacenter Query Load

PocketTrend not only benefits end users, but can also reduce the datacenter's query load by preventing trending search queries from hitting the backend. We evaluate the savings on the datacenter's side for different update mechanisms. In addition to *PT-UpdatesOnly* and *PT-5k* introduced above, we compare PocketTrend to a baseline system (*No-Caching*) with no client-side caching support and to an ideal system (*PT-IdealCache*) which assumes that all users are updated with the freshest trending search content right before they submit their trend-related query.

Figures 10 and 11 show the reduction in the overall search query volume for these four system designs, in the case of U.S. president elections and Boston marathon bombing, respectively. With active updates (*PT-5k*), PocketTrend is able to reduce the datacenter's query volume by 9–17%, depending on the trending event (at the trend's peak time). Note that this reduction of the query vol-

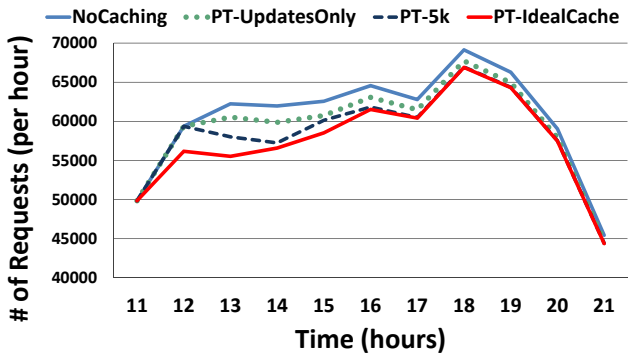


Figure 11: Search query volume reduction with PocketTrend (PT) for different update strategies (Boston marathon bomb).

ume lasts for several hours as the trending event evolves, and is progressively reduced over time as the trending event disappears.

Even more importantly, when passive updates are used (*PT-UpdatesOnly*), PocketTrend is still able to reduce the datacenter's query volume by 7–14%, depending on the trending event. Even though the benefit is lower compared to active updates, the fact that passive updates can achieve similar gains is important as they do not impose as high bandwidth requirements on the datacenter as active updates do (more about this in Section 4.4). However, there can be cases where active update strategies achieve significantly higher query volume reductions. For example, in the Boston marathon event (Figure 11), in the first two hours after the bombing (12pm – 2pm), *PT-5k* is almost twice more efficient than *PT-UpdatesOnly*. We believe that this has to do with the unpredictability of trending events. The Boston marathon bombing event was highly unexpected and dramatic for users, creating this instant spike in user query volume that only active updates can efficiently address initially. On the other hand, the U.S. president election was more expected, and it formed a trending event that slowly developed over time, providing enough time for passive updates to become almost as efficient as active updates[6].

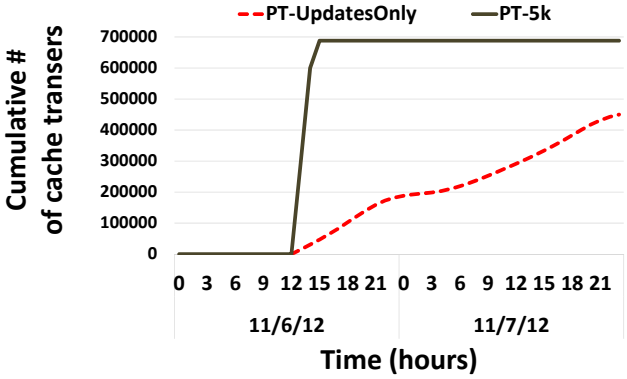
When compared to the ideal cache implementation, both active and passive updates perform remarkably well, with the active strategy having a small edge over the passive one. This is especially noticeable in Figure 10 where *PT-5k* is always within 3% of *PT-IdealCache*. More importantly, *PT-UpdatesOnly* can get at least 63% (usually more) of the benefits provided by *PT-IdealCache*.

Note PocketTrend's 17% reduction of query volume in the case of president elections is lower than the potential reduction of 30% shown in Figure 1. This is due to two reasons. First, it is nearly impossible for PocketTrend to timely predict every single user that will search for the trending event. For some users, predictions cannot be made; some users might also get updated after they have searched for the trending event. Second, Figure 1 presents the results when all trend-related queries are manually selected. In this section, trend-related queries are detected through the PocketTrend's two-step trending content identification algorithm, which might not include every single trend-related query.

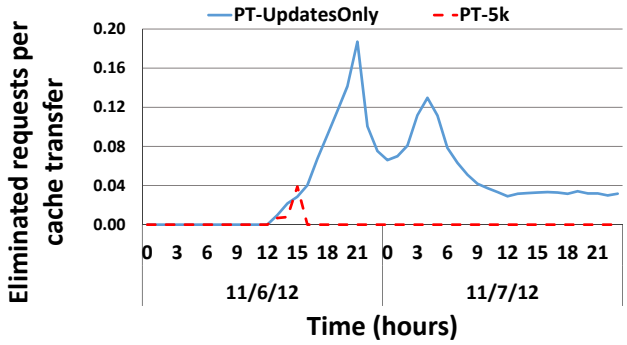
4.4 PocketTrend Overhead

Each time a search request is served locally at the client, one fewer request has to be processed by the datacenter. On the other hand, to avoid such requests the datacenter has to push trending

[6]When active and passive updates are combined, the reduction in the search query volume is only slightly higher than active updates.



(a) Cache transfers



(b) Eliminated datacenter requests per cache transfer

Figure 12: PocketTrend overhead measured as cumulative number of cache transfers (a) and number of eliminated requests to the datacenter per cache transfer (b) (U.S. President Elections).

content to the users, passively or actively. We quantify the overhead of PocketTrend on the datacenter with passive (*PT-UpdatesOnly*) or active (*PT-5k*) updates.

First, we measure the number of trending search content pushes performed by the datacenter. Figure 12(a) shows this for the U.S. president elections. For active updates, 5000 users are updated every minute with the trending search content. A large number of pushes takes place when the trending event is detected. After all users that submitted a search query within 2 hours before the trending event took place have been updated, no more pushes are initiated. With passive updates, pushes occur only for users that came to the search engine for other unrelated topics after the trending event was detected. Because of this, passive update pushes grow slowly over time, and at the end of the event's lifetime, they are half of the active update pushes.

Passive updates are not only fewer, but also more effective. For the same event, we compute the ratio between the number of search requests eliminated from the datacenter and the number of trending content pushes to end users. The higher this ratio is, the lower the overhead from the datacenter's point of view is. As Figure 12(b) shows, passive updates achieve a ratio of almost up to 20%, while active updates remain well below 4%.

Given that the passive and active update strategies provide comparable advantages to end users and to the datacenter, Figure 12 shows that passive updates can be a much more cost-effective approach for the datacenter and the end users. From the datacenter point of view, passive updates do not require any new requests to be

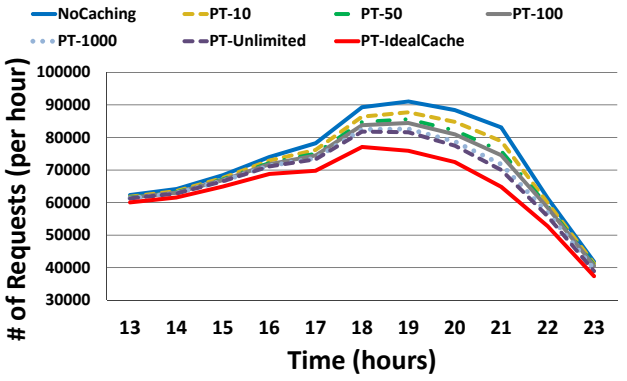


Figure 13: Effect of the size of the pushed trending search content on PocketTrend's query volume reduction (U.S. president elections).

submitted as trending search content is opportunistically pushed to end users, and the number of pushes is halved. From the user point of view, in contrast to active updates where a user's device needs to be explicitly activated to synchronize, passive updates only incur minor battery drain as the user's device radio is already powered up and connected to the search engine.

As we show in more detail in Section 4.5, each push shown in Figure 12(a) corresponds to the datacenter transferring approximately 1MB of additional data. Given the total number of pushes in Figure 12(a), active and passive updates require the datacenter to transfer an additional 680GB and 390GB, respectively. Note that the transfer of this data is distributed over the whole lifetime of a trending event. Given that trending events can last from several hours to tens of hours, this additional bandwidth does not stretch the bandwidth limits of the search backend.

4.5 Trending Content Size Effect

Figures 10 and 11 show the query volume reduction when every single trending search query and search result is pushed. To conclude our tradeoff analysis, we now analyze the PocketTrend's sensitivity on the size of the content pushed to users. We assume PocketTrend employs *UpdatesOnly*; similar results hold for active updates, but are not shown in the interest of space.

Figure 13 shows the overall query volume when the 10, 50, 100, and 1000 most popular trending search queries along with the top 10 most clicked search results for each trending query are pushed (labeled as *PT-** in the figure). As a reference, the results when all trending search queries (Table 3) and their corresponding search results are pushed is also shown (*PT-Unlimited*). Surprisingly, caching 1000 queries (*PT-1000*) provides most of the benefits of pushing every single trending query and search result (*PT-Unlimited*). As a result, PocketTrend can achieve its top performance while pushing a limited number of trending search queries ($\approx$ 1000) and results. Reducing the size of the pushed content to 100 entries significantly reduces the performance gain, while pushing the top 50 entries can halve PocketTrend's performance gain.

Considering that an average search result requires roughly 500 bytes (i.e., URL and data snippet), storing the top 10 search results (i.e., the first page of search results) for a trending query, requires about 5 kB. Hence, pushing 1000 trending search queries corresponds to pushing roughly 5 MB. Note that if PocketTrend were to push all trending search queries and search results (Table 3), more than 135 MB would be used. The size of the trending search content can be further reduced by leveraging data compression techniques. Given that the set of queries and search results pushed to end users relates to the same event, there is significant overlap across the 1000 entries. We experimentally verified that a simple compression algorithm based on bzip2 [27, 31] can achieve a significant compression ratio ranging from 4.5x to 5x. This means that the trending search content can be compressed from 5 MB to just 1 MB. This data size is less than the size of a simple application update on a smartphone today, and it could take place over WiFi links when the cost of cellular bandwidth is a concern.

5. RELATED WORK

Discovering trends in search is a well-studied problem [9]. Commercial search engines already offer products such as Google Trends [6]. Our trend detection approach leverages well-known techniques for trend analysis [20, 24], and is not the main contribution of this work. Instead, the contributions of this work lie on the findings of the search log analysis that show how mobile users search for trending topics. Specifically, the way these findings are leveraged to enable PocketTrend to automatically detect search content related to a trending event, and more importantly to efficiently push this content to users without significantly increasing the datacenter workload is the core contribution of this work.

Previous work on search engine optimization has focused primarily on server-side and client-side caching. In server-side caching [2, 29, 16, 7, 11], search results for popular queries are cached in the search backend. Incoming queries that are in the cache can be answered faster than other queries as there is no need to access the index, and perform any ranking in real-time. The reduction in user response time can be significant when the delay due to search backend's computation is the bottleneck between the user and the datacenter. This tends to be the case when users access search engines through very high speed links (e.g., desktops). Yet, as more and more people use their mobile devices to access search engines through slow cellular links that have high setup times [8], the bottleneck shifts from the datacenter to the cellular links. In this case, the benefit of server-side caching techniques is reduced.

Researchers have been exploiting client-side caching techniques to enable faster user experience in the case of web browsing [19, 30, 5, 17, 28], ad delivery [21], and more recently search result delivery [15]. Koukoumidis et al. [15] showed that a small set of queries and search results represents a significant fraction of the mobile query volume. By analyzing mobile search logs on a daily basis, authors identify the part of the search index that is most often accessed, and use it to create a search engine that lives on the mobile device, and it is able to instantly answer about 60% of the queries an individual user submits. The search results stored on the user's mobile device are only updated nightly when the phone is charging and connected to WiFi. Even though quite effective, this work is only limited to relatively static search content that can be predicted through periodic search log mining. This system cannot address unpredicted query volume spikes due to trending events taking place throughout the day as the necessary trending search content will not be available on the users' mobile devices. Moreover, the authors blindly update every user assuming that the updates take place during off-peak times (e.g., overnight), and over WiFi connections. This approach, in the case of trending events, would require the datacenter to simultaneously update hundreds of millions of users during peak time, creating a larger problem for the search backend than the one we solve.

Conversely, PocketTrend automatically detects trending events in real time, and identifies the search content associated to these events. The trending search content is then intelligently pushed either actively to a subset of users that with high probability will

search for the trending event, or passively to any user that reaches the search engine after a trending event has been detected and is still active. In that way, PocketTrend addresses trending query volume spikes, without stressing even further the search engine's backend.

Unpredicted workload spikes is a reality for most web services [24], and are not limited to search engines [26]. As a result, there have been a lot of efforts to address these challenges directly at the networking level within datacenters [10, 22, 1]. These techniques are orthogonal to our work.

6. CONCLUSION

We have analyzed 21 million mobile search queries to understand how trending search topics are formed, and how they evolve over time. Based on our findings, we have designed and evaluated PocketTrend, a new architecture that is capable of servicing user queries related to trending events locally, improving both the user experience and the datacenter query load. Our evaluation using real mobile search logs, showed that in the presence of a trending event, up to 13%–17% of the overall traffic can be eliminated from the datacenter, impacting as many as 19% of all users.

Acknowledgements

We thank the reviewers for their valuable suggestions. Gennady Pekhimenko is supported by a Microsoft Research Fellowship and a Qualcomm Innovation Fellowship.

7. REFERENCES

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *Proc. of SIGCOMM*, pages 63–74, 2010.

[2] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *Proc. of SIGIR*, pages 183–190, 2007.

[3] R. Baeza-yates, F. Junqueira, V. Plachouras, and H. F. Witschel. Admission policies for caches of search engine results. In *Proc. of the 14th String Processing and Information Retrieval Symposium*, volume 4726 of *LNCS*, pages 74–85, 2007.

[4] R. Baeza-Yates and F. Saint-Jean. A three level search engine index based in query log distribution. In *Proc. of the 10th String Processing and Information Retrieval Symposium*, volume 2857 of *LNCS*, pages 56–65, 2003.

[5] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc. of IMC*, pages 280–293. ACM, 2009.

[6] H. Choi and H. Varian. Predicting the present with google trends. *Economic Record*, 88:2–9, 2012.

[7] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24(1):51–78, Jan. 2006.

[8] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *Proc. of MobiSys*, 2010.

[9] N. G. Golbandi, L. K. Katzir, Y. K. Koren, and R. L. Lempel. Expediting Search Trend Detection via Prediction of Query Counts. In *Proc. of WSDM*, 2013.

[10] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proc. of SIGCOMM*, pages 51–62, 2009.

[11] S. Jonassen, B. B. Cambazoglu, and F. Silvestri. Prefetching Query Results and Its Impact on Search Engines. In *Proc. of SIGIR*, 2012.

[12] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[13] M. Kamvar and S. Baluja. A large scale study of wireless search behavior: Google mobile search. In *CHI*, 2006.

[14] M. Kamvar, M. Kellar, R. Patel, and Y. Xu. Computers and iphones and mobile phones, oh my! In *WWW*, 2009.

[15] E. Koukoumidis, D. Lymberopoulos, K. Strauss, J. Liu, and D. Burger. Pocket Cloudlets. In *ASPLOS*, 2011.

[16] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *Proc. of WWW*, pages 19–28, 2003.

[17] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas. PocketWeb: instant web browsing for mobile devices. In *ASPLOS*, pages 1–12, 2012.

[18] H. Ma and B. Wang. User-aware Caching and Prefetching Query Results in Web Search Engines. In *Proc. of SIGIR*, 2012.

[19] E. P. Markatos and C. E. Chronaki. A top-10 approach to prefetching on the web. In *Proc. of INET*, 1998.

[20] M. Mathioudakis and N. Koudas. TwitterMonitor: trend detection over the twitter stream. In *Proc. of SIGMOD*, pages 1155–1158, 2010.

[21] P. Mohan, S. Nath, and O. Riva. Prefetching mobile ads: can advertising systems afford it? In *Proc. of EuroSys*, pages 267–280, 2013.

[22] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. I. T. Rowstron. Everest: Scaling Down Peak Loads Through I/O Off-Loading. In *OSDI*, pages 15–28, 2008.

[23] Pyevolve. Machine Learning - Text feature extraction (tf-idf). http://pyevolve.sourceforge.net/wordpress/?p=1589, 2014.

[24] K. Radinsky and E. Horvitz. Mining the Web to Predict Future Events. In *Proc. of WSDM*, 2013.

[25] A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011.

[26] A. Saha and V. Sindhwani. Learning Evolving and Emerging Topics in Social Media: A Dynamic Nmf Approach with Temporal Regularization. In *Proc. of WSDM*, 2012.

[27] J. Seward. Bzip2 V. 1.0.6. http://www.bzip.org/, 2010.

[28] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. How far can client-only solutions go for mobile browser speed? In *Proc. of WWW*, pages 31–40, 2012.

[29] Y. Xie and D. R. O'Hallaron. Locality in search engine queries and its implications for caching. In *INFOCOM*, 2002.

[30] L. Yin and G. Cao. Adaptive power-aware prefetch in wireless networks. *IEEE Trans. on Wireless Comms*, 2004.

[31] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Trans. Inf. Theory*, 1977.