Computer Networks 52 (2008) 3248–3257

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

The eternal sunshine of the sketch data structure

Xenofontas Dimitropoulos *, Marc Stoecklin, Paul Hurley, Andreas Kind

IBM Zurich Research Laboratory, Saumerstrasse 4, CH-8803 Zurich, Ruschlikon, Switzerland

A R T I C L E   I N F O

Article history:
Received 14 June 2007
Received in revised form 16 May 2008
Accepted 13 August 2008
Available online 27 August 2008

Responsible Editor T. Tugcu

Keywords:
Network monitoring
Data streaming
Sketch data structure

A B S T R A C T

In the past years there has been significant research on developing compact data structures for summarizing large data streams. A family of such data structures is the so-called *sketches*. Sketches bear similarities to the well-known Bloom filters [B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of ACM, 13 (7) (1970), 422–426] and employ hashing techniques to approximate the count associated with an arbitrary key in a data stream using fixed memory resources. One limitation of sketches is that when used for summarizing long data streams, they gradually saturate, resulting in a potentially large error on estimated key counts. In this work, we introduce two techniques to address this problem based on the observation that real-world data streams often have many transient keys that appear for short time periods and do not re-appear later on. After entering the data structure, these keys contribute to hashing collisions and thus reduce the estimation accuracy of sketches. Our techniques use a limited amount of additional memory to detect transient keys and to periodically remove their hashed values from the sketch. In this manner the number of keys hashed into a sketch decreases, and as a result the frequency of hashing collisions and the estimation error are reduced. Our first technique in effect slows down the saturation process of a sketch, whereas our second technique completely prevents a sketch from saturating.[1] We demonstrate the performance improvements of our techniques analytically as well as experimentally. Our evaluation results using real network traffic traces show a reduction in the collision rate ranging between 26.1% and 98.2% and even higher savings in terms of estimation accuracy compared to a state-of-the-art sketch data structure. To our knowledge this is the first work to look into the problem of improving the accuracy of sketches by mitigating their saturation process.

1. Introduction

In the past several years there has been significant research on developing data structures for summarizing massive data streams and on computing desired statistics from the summaries created. One statistic that is commonly sought is the total count associated with a key in a data stream, i.e., the sum of the values that have appeared for a key. Sketches are summary data structures that can estimate the count associated with a key, a process that is also called answering *point queries*. In networking, sketches have known applications in estimating the size of the largest traffic flows in routers [5] and NetFlow/IPFIX collectors [9], in detecting changes in traffic streams [10,16], in adaptive traffic-sampling techniques [12], and in worm fingerprinting [17]. More generally, sketches find applications in systems that require online processing of large data sets.

Sketches are a family of data structures that use the same underlying hashing scheme for summarizing data. They differ in how they update hash buckets and use hashed data to derive estimates. Among the different

* Corresponding author. Tel.: +41 44 724 8768.
E-mail addresses: xed@zurich.ibm.com (X. Dimitropoulos), mtc@zurich.ibm.com (M. Stoecklin), pah@zurich.ibm.com (P. Hurley), ank@zurich.ibm.com (A. Kind).

[1] The phrase "eternal sunshine" in the title reflects that our techniques mitigate or halt the saturation process of a sketch.

1389-1286/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2008.08.014

sketches, the one with the best time and space bounds is the so-called *count-min* sketch [4]. Sketches bear similarities to the well-known Bloom filters [2] in that both employ hashing to summarize data, have fixed memory requirements, and provide approximate answers to the queries they are designed for.

An inherent problem with sketches is that keys may collide, namely, hash to the same bucket, producing errors in the estimated counts. This particularly affects long data streams, for which sketches gradually saturate, leading to more collisions and to lower estimation accuracy. In this work we introduce two techniques to address this problem. Our techniques are based on the observation that for real-world data streams, the saturation of sketches is driven by a large number of transient keys that are hashed into the data structure and produce many collisions. These transient keys often are not interesting either because they have a small size or because they become inactive after a certain point in time. The techniques effectively detect such keys and remove their hashed values from the sketch, decreasing in this way the number of collisions and the estimation error. The first technique uses a small additional memory to detect certain transient keys and to remove their hashed values from the sketch. Transient keys that are not detected remain in the sketch, and thus, sketch saturation slows down but does not halt completely. The second technique uses a larger additional memory to effectively halt the saturation process by posing an upper bound on the number of keys that are present in the data structure at any given point in time.

Our techniques are useful for applications that are not interested in transient keys, but rather focus on the active or recent keys of a data stream. In the context of network monitoring and management, a number of such applications exist that, for example, monitor the active flows in a network, identify important events, like anomalies, and manipulate monitored active flows, namely, for traffic engineering purposes.

We describe and analyze our techniques with respect to the state-of-the-art count-min sketch.[2] Using traffic traces from a large enterprise network, we illustrate that, compared with the count-min sketch, our techniques result in lower collision rate and increased estimation accuracy.

We structure the remainder of this paper as follows: in the next section we provide background information on sketches and introduce our notation. Then, in Section 3 we summarize related research. In Section 4 we introduce our two techniques. We evaluate the performance of our techniques and compare it with that of existing schemes in Section 5. Finally, we conclude our paper and outline future directions in Section 6.

2. Preliminaries

In this section we first outline the data-streaming model that forms the input to a sketch data structure, and then provide a detailed description of the count-min sketch.

2.1. Data-streaming model

A data stream I of running length n is a sequence of n tuples. The t-th tuple is denoted as (k_t, u_t), where k_t is a key used for hashing and u_t is a value associated with the key:

$$I = (k_0, u_0), (k_1, u_1), \ldots, (k_t, u_t), \ldots, (k_{n-1}, u_{n-1}).$$

The value u_t is a positive number, which results in monotonically increasing key counts. This data-streaming model is called the cash register model [15].

2.2. The count-min sketch

The count-min sketch i s a two-dimensional array $T[i][j]$ with d rows, $i = 1, \ldots, d$, and w columns, $j = 1, \ldots, w$. Each of the $d \times w$ array buckets is a counter that is initially set to zero. The rows of T are associated with d pairwise independent hash functions $h_1, \ldots, h_d$ from keys to $1, \ldots, w$. In Fig. 1 we illustrate the count-min sketch data structure.

Update procedure. An incoming key k_t is hashed with each of the hash functions $h_1, \ldots, h_d$, and the value u_t is added to the counters $T[i][h_i(k_t)], i = 1, \ldots, d$.

Estimation procedure. Given a query for key k_t, the count-min sketch returns the minimum of the counters to which k_t hashes, so that the estimated count for a key k_t becomes $\min_{i=1,\ldots,d} T[i][h_i(k_t)]$.

Discussion. First, note that multiple keys may hash to the same bucket and thus, the count of a bucket may *overestimate* the true size of a key. For this reason the estimation procedure returns the minimum value of the counters a key is hashed to. Assume for a moment that the count-min sketch has a single row, i.e., $d = 1$. If two keys hash to the same counter, the counter will overestimate the count of the two keys. By adding more hash buckets in each row, i.e., increasing the number of columns w, it becomes less likely that two keys collide and thus, the number of collisions and the resulting *expected error* decrease. Accordingly, the parameter w primarily manipulates the expected error of the count-min sketch. If we increase the number of rows d of the sketch, then the probability that a given key has a large error, i.e., collides with many other keys, decreases. This is because as we increase the number of rows d, it becomes less likely that a given key collides with *many* other keys in all of the d buckets it is hashed to. Therefore, the parameter d mainly controls

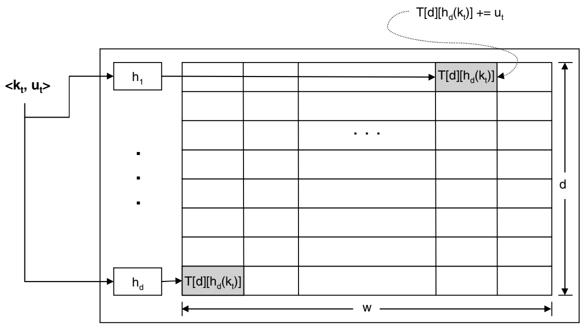

Fig. 1. The count-min sketch data structure.

[2] Though we focus on the count-min sketch, our techniques are generic and can be used with other sketch variants as well.