

# TCAM-conscious Algorithms for Data Streams \*

Nagender Bandi    Ahmed Metwally    Divyakant Agrawal    Amr El Abbadi  
Department of Computer Science  
University of California, Santa Barbara  
{nagender, metwally, agrawal, amr}@cs.ucsb.edu

## Abstract

Recently, there has been significant interest in developing space and time efficient solutions for answering continuous summarization queries over data streams. While these techniques are evaluated in a standard CPU setting, many of their applications such as click-fraud detection, and network-traffic summarization typically execute on special networking architectures called Network Processing Units (NPU). These NPUs interface with special kind of associative memories known as the Ternary Content Addressable Memories (TCAMs). In this paper, we describe how the integrated architecture of NPU and TCAMs can be exploited towards achieving the goal of developing high-speed stream summarization solutions. We analyze popular solutions for the frequent elements problem in data stream, discuss the bottleneck issues and motivate how TCAMs can help alleviate these bottlenecks. A preliminary evaluation on an NPU platform reveals the performance gains of the TCAM-conscious techniques over software implementations.

## 1 Introduction

During the last decade, researchers in the data stream community have presented diverse solutions for answering continuous summarization queries over data streams. Some of the popular summarization problems are finding the most frequent elements [9, 5, 10], and quantile summarization [7] of a data stream. Given a continuous stream, the frequent elements problem involves finding the data elements whose frequency of occurrence in the stream is greater than a user-defined threshold. Quantile summarization provides statistics such as the median and histograms. These problems have a wide range of applications such as click fraud detection, network-traffic summarization, and online analysis of stock market data. For example, a network administrator can issue the frequent elements query to find all the network flows that contribute more than 1% of network link capacity at a particular router. Similarly, a stock market analyst can issue a quantile query to get approximate representation of the stock transactions on a particular day.

As a data stream algorithm operates over an infinite stream of elements where an element is accessed only once, they are designed to be efficient in terms of both space and time complexity. Typically, these techniques maintain data structures such as binary search trees, heaps, and priority queues that aid in executing only a bounded number of operations per stream element. However these bounds are typically logarithmic or amortized  $O(1)$  cost (with a relatively large constant) in terms of the stream size or the data space. Further, these data structures are built using pointers, which is known to be inefficient in terms of memory access cost.

We observe that many applications such as click-fraud detection and network-traffic summarization execute on network routers that operate on non-conventional hardware called the Network Processing Units (NPUs). While the conventional processor architectures have not changed much over the last decade, networking hardware has evolved significantly. In particular, the memory hierarchy of NPUs features a Ternary Content Addressable Memory (TCAM) that enables constant time lookup for an element among a large collection of elements. Off-the-shelf TCAMs provide *constant time* searches at speeds of 100 million searches per second and provide this throughput irrespective of the size of data repository. In this paper, we describe how NPUs and TCAMs can be exploited to provide faster solutions for data stream problems. In particular, we describe how some of the well known algorithms proposed for conventional processor architectures can be adapted to the Associative Memory model of a TCAM towards providing highly efficient implementations.

## 2 Advanced Networking Architecture

As the traditional hardware was not built to be optimal for networking applications, custom application-specific hardware was built that can operate at multi-gigabit speeds [1]. However such systems offer limited configurability and software programmability thereby limiting the reuse of existing systems. In order to enable re-usability, high-performance microprocessors called the Network Processing Unit (NPU) have been developed with a very flexible software programming capability.

\*This work is supported by NSF grants IIS 02-23022 and CF 04-23336

A typical network processor, such as the Intel IXP2800 [1], has one control plane processor (CPP) and 16 data plane processors on a single die. The CPP is a standard 32-bit low-power high-performance Xscale processor that runs an embedded operating system such as Monta Vista Linux. The data plane processor, also referred to as a micro-engine (ME), is a 32-bit low-power RISC processor with 8 thread contexts thus providing the IXP2800 NPU with 128 (16\*8) parallel threads of raw computation power. One of the main difference between a conventional multiple processor architecture and the NPU is the method of communication between the MEs. Apart from the traditional method of shared bus and memory sharing, the MEs are organized in a chain-like fashion that is similar to the classical ring topology. Each ME can communicate with two of its neighbors through shared registers. This makes the NPU ideal for streaming applications as it allows an application to be componentised across multiple processors (MEs and their threads) in a pipe-lined fashion.

We observe that the integrated setup of NPU and TCAMs has been used predominantly for building routing applications. Recently there has been significant interest in exploiting NPUs and TCAMs for developing non-routing applications, e.g. giga-bit rate intrusion detection[6], giga-bit rate pattern-matching [11], and relational database joins [3]. However, this setup has not been exploited for developing efficient traffic summarization solutions, a popular data stream application. Using the state-of-the-art NPU platform, the IXDP2801, which feature an IXP2800 NPU and using TCAMs from Integrated Devices Technology, we built an integrated setup for developing our solutions. In the rest of this paper, we discuss how we use our setup towards developing and evaluating efficient summarization techniques on data streams.

### 3 Frequency Counting

Given a continuous stream of elements, the frequent elements query lists all the elements that have a frequency of more than a certain threshold. Another closely related query is the *top-k* query that lists the k most frequent elements. The frequent elements problem can be formally stated as follows: given an alphabet,  $A$ , a frequent element,  $e_i$ , is an element whose frequency,  $f_i$ , in a stream  $S$  of a given size  $N$ , exceeds a user-specified support  $\phi N$ , where  $0 \leq \phi \leq 1$ ; whereas the top-k elements are the k elements with highest frequencies.

The solutions proposed for these problems can be broadly classified into counter-based techniques and sketch-based techniques. Counter-based techniques [8, 9, 10] keep an individual counter for each element in the monitored set, a subset of  $A$ . The counter of a monitored element,  $e_i$  is updated when  $e_i$  occurs in the stream. On the other hand, sketch-based techniques [4, 5] do not monitor a subset of elements. They provide, with less stringent guarantees, fre-

quency estimation for all elements using an array of counters. A given counter is updated for all elements that hash to this counter, thereby maintaining frequency count information for a subset of the alphabet. Furthermore, each element is hashed to update several counters. For each element, the counters it hashes to are queried for the element frequency with less accuracy, due to hashing collisions.

Sketch-based techniques are not only known to have a large space complexity but also provide only probabilistic information about individual elements' frequencies [10]. Furthermore, they monitor all elements in the data stream where a hit entails expensive calculations thus making the sketch-based solutions not so ideal for faster implementations. Therefore, we consider only counter-based solutions in this paper. Although all the counter-based solutions for finding the frequent elements were designed to be both space and compute efficient, they incur a logarithmic or amortized constant number of operations (in terms of stream size) per stream element [10]. The data structures used in these techniques are typically implemented as trees, priority queues, heaps, hash tables and operations over these data structures contribute for the above mentioned cost. These data structures are used to efficiently store the data stream summary and facilitate lookups for incoming stream elements in the current summary. On the other hand, we observe that TCAMS provide constant time lookups over large amount of data. These TCAMs are readily available on an NPU where several applications of frequent elements problem are typically used. In the rest of this paper, we explore whether this constant time lookup feature of a TCAM can be exploited by our algorithms for faster implementations.

We analyze two of the most-efficient counter-based techniques, the Lossy Counting [9] and Space Saving [10]. Given a data stream of size  $N$  and a user-specified error threshold  $\epsilon$ , Lossy Counting returns all the most frequent elements with an error bound of  $\epsilon * N$ . This technique has a space complexity of  $O(1/\epsilon \log(N * \epsilon))$ . Space Saving on the other hand has a rigid space bound of  $1/\epsilon$  and answers the frequent elements query with an error threshold of  $\epsilon$ . While Lossy Counting has relatively higher space complexity, Space Saving requires the summary structure to be maintained as a priority queue, which is computationally expensive.

We describe briefly how TCAMs can be exploited towards adapting Lossy Counting to an NPU/TCAM setting for faster performance. In [2], we discuss in detail how both techniques can be adapted to an NPU/TCAM setting. The Lossy Counting technique [9] divides the stream into rounds of length  $\frac{1}{\epsilon}$ . Each round is associated with a round number  $r$  which is set to  $\lceil \epsilon N \rceil$ . The algorithm maintains a data structure  $G$  that contains entries of the form  $\langle e, f, \Delta \rangle$  where  $e$  is an element,  $f$  is its estimated frequency, and  $\Delta$  is the maximum error in the frequency estimation. All the

elements in  $G$  satisfy the inequality  $f \leq f_e \leq f + \Delta$ , where  $f_e$  is the exact frequency of the element  $e$ . In every round,  $r$ , when an element,  $e$ , is observed, its frequency in  $G$  is incremented. If  $e$  is not monitored in  $G$ , a new entry  $\langle e, 1, r \rangle$  is added to  $G$ . At the end of each round,  $r$ , all element satisfying  $(f + \Delta) \leq r$  are deleted. Intuitively, this step which we refer to as *zeroing step* removes all the elements with lower frequencies. This algorithm has a proven upper bound on space usage of  $O(1/\epsilon \log(\epsilon N))$ , and using a smart optimization, every stream element incurs an amortized cost of  $O(\log(1/\epsilon))$  [9].

Lossy Counting is very well suited to be adapted to the TCAM model. When a new stream element  $e'$  arrives, it is matched in constant time against all the  $e$ 's stored inside the TCAM. If there is a match, the counter's frequency can be incremented appropriately. Otherwise, a TCAM entry corresponding to the new counter is added to the table. Initially, we can assume that the new counter is allocated a TCAM entry towards the end of the table stored in the TCAM. One important issue that needs to be addressed in this adaptation is the garbage collection of TCAM space. At the end of each round, several counters will be zeroed. Thus, those TCAM entries should be available for future allocation, and should no longer be valid in future searches. This can be done by disabling the VALID bit associated with each TCAM entry in constant time. When allocating a new entry in the next round, the such first entry is returned in constant time using the LEARN command. Assuming that all the TCAM operations have similar cost, any element incurs a total cost of 3 to 5 TCAM operations.

## 4 Experimental Evaluation

In this section, we refer to the TCAM-adapted versions of Lossy Counting and Space Saving as TCAM-conscious algorithms and present a preliminary evaluation. Figure 1 present the performance of TCAM-conscious Lossy Counting, TCAM-conscious Space Saving and their software counterparts with zipfian factors varying from 0 to 3. Here after, we refer to the TCAM-conscious Lossy Counting, TCAM-conscious Space Saving, software Lossy Counting and software Space Saving as TLC, TSS, SLC and SSS respectively. These results show that the TCAM versions of both techniques perform at least 3 times faster than their software counterparts. This is despite the fact that the software techniques are evaluated on a faster and architecturally superior Pentium 4 processor.

While TSS's performance does not change much with zipfian factor, TLC shows significant difference in processing cost. At lower zipfian factors, the data becomes more uniform. Thus, the frequency of each element of the alphabet is small compared to the size of dataset. Hence, it incurs more operations per element than at higher zipfian factors

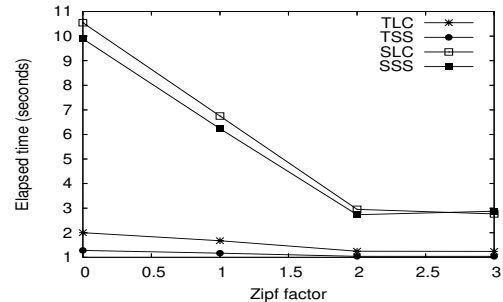


Figure 1. Processing time

## 5 Conclusion

In this paper, we explored the frequent elements problem in data streams from a networking systems perspective. We proposed to exploit TCAMs, a special kind of memories found in NPUs, for providing efficient implementations for the frequent elements problem. We proposed two solutions, the TCAM-conscious Lossy Counting and TCAM-conscious Space Saving, implemented both techniques on an Intel's IXDP2801 NPU platform, and evaluated them over synthetic zipfian data. Experimental results show that the TCAM-conscious techniques are at least 3 times faster than their software counter-parts. We are currently exploring if other complex features of TCAM, such as the range matching and sorting can help provide faster solutions for the data stream problems in networking environments.

## References

- [1] Intel network processing units. [www.intel.com](http://www.intel.com).
- [2] N. Bandi, A. Metwally, D. Agrawal, and A. El Abbadi. Tcam-conscious algorithms for data streams. Technical report, Department of Computer Science, University of California Santa Barbara, 2006.
- [3] N. Bandi, S. Schnieder, D. Agrawal, and A. El Abbadi. Hardware acceleration of database operations using content-addressable memories. In *DaMoN*, 2005.
- [4] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *ICALP'02*, pages 693–703, 2002.
- [5] G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. In *PODS '03*, pages 296–306, 2003.
- [6] R. H. K. Fang Yu. Efficient Multi-Match Packet Classification with TCAM. In *IEEE Hot Interconnects*, 2004.
- [7] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD Conference*, 2001.
- [8] J. Hershberger, N. Shrivastava, S. Suri, and C. D. Tóth. Adaptive spatial partitioning for multidimensional data streams. In *ISAAC*, pages 522–533, 2004.
- [9] G. S. Manku and R. Motwani. Approximate frequency counts over data streams, 2002.
- [10] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, pages 398–412, 2005.
- [11] F. Yu, R. H. Katz, and T. V. Lakshman. Gigabit rate packet pattern-matching using tcam. In *ICNP '04*, pages 174–183, 2004.