Project report

Software based measurement sketch using DPDK pipeline

Jigar Makwana
University of Massachusetts - Lowell

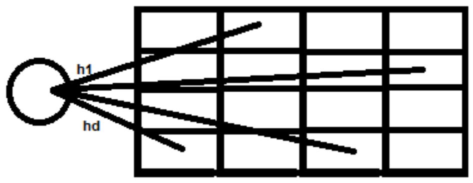
Abstract:
The Count-Min (CM) Sketch is a compact summary data structure capable of representing a high-dimensional vector and answering queries on this vector, in particular point queries and dot product queries, with strong accuracy guarantees. Such queries are at the core of many computations, so the structure can be used in order to answer a variety of other queries, such as frequent items (heavy hitters), quantile finding, join size estimation, and more. Count-Min sketch is an efficient approximate query tool for data stream. In this paper, we address how to further improve its point query performance. Firstly, we modify the estimation method under cash register model. Our method will relieve error propagation. Secondly, we find better method under turnstile model and prove that our method is more efficient than that Count-Min sketch. These conclusions are well supported by experimental results.

Introduction:
With growing concerns of the cost and management difficulty of hardware network devices (switches, middleboxes), there is a new trend of moving some network functions to software. Network switches and routers in modern datacenters, enterprises, and service-provider networks perform many tasks in addition to standard packet forwarding. In the DPDK network data typically comes at very high rate. High rate means it stresses communication and computing infrastructure, so it may be hard to transmit the entire input to the program; hard to compute sophistications on large pieces of the input at the rate it is presented and hard to store all of them in memory or disk. In other words, it make answer queries exactly impossible.

DPDK:
DPDK has sets of libraries and drivers that provides a simple, complete framework for fast packet processing in data plane applications. These libraries are used for creation of an Environment Abstraction Layer, which may be specific to a mode of the Intel® architecture (32-bit or 64-bit), Linux user space compilers or a specific platform. These environments are build with the use of make files and configuration files. Once the Environment Abstraction Layer library is created, the user can link with the library to create their own applications. There are also other libraries outside of Environment Abstraction Layer which have the Hash, Longest Prefix Match and rings libraries.
Some of the main libraries are multicore framework, ring buffers, poll-mode drivers for networking, huge page memory, crypto and eventdev.
The ring buffer provides a lockless multi-producer, multi-consumer FIFO API in a finite size table. It is easier to implement, adapted to bulk operations and faster. A ring is used by the Memory Pool Manager and can be used as a general communication between cores and/or execution blocks connected on a logical core. It also has some disadvantages like size is fix and more rings costs more in terms of memory than a linked list queue.
The Memory Pool Manager is responsible for allocating pools of objects in memory. A pool is identified by name and uses a ring to store free objects. It also provides services like a per-core object cache and an alignment helper to ensure that objects are padded to spread them equally on all RAM channels.
A Poll Mode Driver for 1GbEthernet, 10GbEthernet and 40GbEthernet, and para virtualized virtio Ethernet controllers which are designed to work without asynchronous,

interrupt-based signaling mechanisms. It uses the BSD driver running in user space and provides APIs which helps to configure the devices and their respective queues. A Poll Mode Driver also accesses the RX and TX descriptors directly without any interrupts to quickly receive, process and deliver packets in the user's application.
Hash and Longest Prefix Match libraries to support the corresponding packet forwarding algorithms. Hash Library is used to creating hash table for fast lookup. The hash table is a data structure optimized for searching through a set of entries that are each identified by a unique key.
Hugepage support is required for the large memory pool allocation used for packet buffers. Using Hugepages performance is increased since fewer pages are needed, and therefore less Translation Lookaside Buffers, which reduce the time it takes to translate a virtual page address to a physical page address. Without hugepages, high TLB miss rates would occur with the standard 4k page size, slowing performance. For 64-bit applications, it is recommended to use 1 GB hugepages if the platform supports them.
Using these libraries, we can receive and send packets within the minimum number of CPU cycles, develop fast packet capture algorithms, run third-party fast path stacks.

Count–min sketch:
count–min sketch also known as CM sketch is a probabilistic data structure that supports as a frequency table of events in a stream of data. It does that by using hash functions to capture events to frequencies, but it uses only sub-linear space to avoid collisions. The count–min sketch was invented in 2003 by Graham Cormode and S. Muthu Muthukrishnan.
The CM sketch is simply an array of counters of width w and depth d, CM[1, 1] . . . CM[d, w]. Each entry of the array is initially zero. Additionally, d hash functions, in our case we used 3 hash functions.
h1 . . . hd : {1 . . . n} → {1 . . . w}
hash functions are chosen regularly at random from. The data structure is represented by wd counters and d hash functions.



Design and Implementation of the count-min algorithm:
There are several approaches to implement the count-min algorithm the way that we have implemented the algorithm will be as follows.

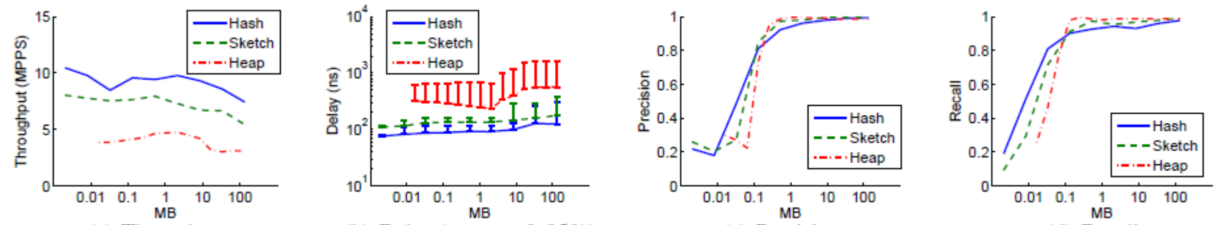
Step 1: Get the 5 tuples from the packet header. (In our case we are dealing with IPv4 and use the source and destination IP only)
Step 2: Put the source and destination IP values in the 3 different hash function and get 3 different hash values. (which will be a decimal number between 0 to 1024).
 - In the Conversion process take the IP (eg. 192.34.54.34) and convert that to decimal d, which will be a very large number (d>>>1024) (eg. 34545675445)
 - Do the mode operation of d with 1024 which will give the number less that 1024 (d % 1024) (0<hv1,hv2,hv3<1024)
Step 3:Map that hash value with the column in the hash table.
 (eg. For the first hash function hv1 = 25 ,then the value in the hash table will be arr[0][25])
Step 4:Increment the count at the position in the hash table.
 (eg. The count at the position of arr[0][25] will get incremented by 1.)
Step 5: Do the same procedure for hv2 and hv3.

Step 6: Find the minimum of the three hash counts.
Step 7: Apply the condition and get the IP(source and destination) which hits the most.

Here in order to implement the algorithm we have taken 3 different hash values and computed the minimum value of the three. We can take as many hash values possible in order to implement the algorithm, which will add the precision in finding the heavy hitter.

Performance:
We evaluated the heavy hitter detection on a; Xeon E5- 2650 processor with 12 cores, 256 KB L2 cache per core,25 MB shared L3 cache, and

Table 1: A survey of proposed measurement solutions



(a) Throughput (b) Delay (average & 95%) (c) Precision (d) Recall

32 GB memory. We send traffic with a 100 G network interface card by using a modified version of Click modular router with DPDK.
We implemented three algorithms: a simple hash table, a Count-Min sketch, and a heap-based solution.
The simple hash table maintains one counter at each entry, and increments the counter based on the hash value of each packet. The counters may be inaccurate when multiple flows are hashed in the same bin. The Count-Min sketch is similar to the simple hash table but uses multiple hash functions for each packet.

We first compare the throughput and delay of the three algorithms with different sizes of their data structures. Figure (a) shows that the throughput for the simple hash function is on average 38% higher than the Count-Min sketch and 110% higher than the heap-based approach. This is consistent
to the average latency comparison in Figure (b). Sketch has worse performance than Hash because it computes 3 hash functions per packet and accesses 3 random memory locations. As per the results Sketch
has worse performance than Hash because it computes 3 hash functions per packet and accesses 3 random memory locations. Similarly, Heap has the worst performance than
the other two because it takes multiple memory accesses to navigate and maintain the heap data structure.

With Hash and Sketch, there is a large interval where increasing the size does not affect the average latency (up until 20 MB) (Figure (b)). This interval happens because L2 and L3 memory cache the flow counters very effectively and most of the popular counters are served by these two caches.
In contrast, Heap has decreasing latency with larger sizes. This is because with larger sizes, the heavy hitters all fall in the leaves in the min-heap. Therefore, we do not need to re-arrange the heap much as packets pass by.
Figure (b) also shows the 95-percentile latency for the three approaches. Tail latency is critical for the stability of packet processing performance. This is because with large
latency variance, the NIC may maintain longer queues at times leading to higher chances of packet losses. The tail latency in Figure (b) has a jump between the 10 MB to 30 MB, which overlaps with the size of L3. Heap has increasing tail latency with larger sizes. In fact, the tail latency grows proportionally with the depth of

the heap. This is because at the tail the packets often require reorganizing the heap, which takes longer time with larger heaps.

Conclusion:

This project re-evaluates the measurement algorithms used for reducing memory usage in the new context of software servers. Our experiments and analysis show that a simple hash table actually works fine for a variety of measurement scenarios.

Acknowledgement:

We sincerely thank the Professor Yan Luo and course T.A. Xiaoban Wu for the support that we have had during the entire project execution.

Github Link:

https://github.com/jigar-makwana/DPDK

Reference:

[1] T. Benson, A. Akella, and D. A. Maltz. Network Traffic
Characteristics of Data Centers in the Wild. In IMC, 2010.

[2] G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and its Applications. Journal of Algorithms, 55(1), 2005.

[3] Omid Alipourfard, Masoud Moshref, Minlan Yu Re-evaluating Measurement Algorithms in Software

[4] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz,R. Kern, H. Kumar, M. Zikos, H. Wu, C. Kim, and N. Karri.Ananta: Cloud Scale Load Balancing. In SIGCOMM, 2013.

[5] A. Kumar, M. Sung, J. J. Xu, and J. Wang. Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution. In SIGMETRICS, 2004.

[6] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca. Planck: Millisecond-scale Monitoring and Control for Commodity Networks. In SIGCOMM, 2014