Computer Networks 66 (2014) 52–67

Contents lists available at ScienceDirect

Computer Networks

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

Recursive design of hardware priority queues ☆

Y. Afek [a,*,1], A. Bremler-Barr [b,2], L. Schiff [a,1,2]

[a] Tel Aviv University, Tel Aviv, Israel
[b] The Interdisciplinary Center, Hertzelia, Israel

ARTICLE INFO

Keywords:
Sorting
TCAM
Priority Queue
WFQ

ABSTRACT

A recursive and fast construction of an n-element priority queue from exponentially smaller hardware priority queues and size n RAM is presented. All priority queue implementations to date require either $O(\log n)$ instructions per operation or, exponential (with key size) space or, expensive special hardware whose cost and latency dramatically increases with the priority queue size. Hence constructing a priority queue (PQ) from considerably smaller hardware priority queues (which are also much faster) while maintaining the $O(1)$ steps per PQ operation is critical. Here we present such an acceleration technique called the Power Priority Queue (PPQ) technique. Specifically, an n-element PPQ is constructed from $2k-1$ primitive priority queues of size $\sqrt[k]{n}$ $(k=2,3,\ldots)$ and a RAM of size n, where the throughput of the construct beats that of a single, size n primitive hardware priority queue. For example an n-element PQ can be constructed from either three $\sqrt{n}$ or five $\sqrt[3]{n}$ primitive H/W priority queues.

Applying our technique to a TCAM based priority queue, results in TCAM-PPQ, a scalable perfect line rate fair queuing of millions of concurrent connections at speeds of 100 Gbps. This demonstrates the benefits of our scheme; when used with hardware TCAM. We expect similar results with systolic arrays, shift-registers and similar technologies.

As a byproduct of our technique we present an $O(n)$ time sorting algorithm in a system equipped with a $O(w\sqrt{n})$ entries TCAM, where here n is the number of items, and w is the maximum number of bits required to represent an item, improving on a previous result that used an $\Omega(n)$ entries TCAM. Finally, we provide a lower bound on the time complexity of sorting n-element with TCAM of size $O(n)$ that matches our TCAM based sorting algorithm.

1. Introduction

A priority queue (PQ) is a data structure in which each element has a priority and a dequeue operation removes and returns the highest priority element in the queue. PQs are the most basic component for scheduling, mostly used in routers, event driven simulators and is also useful in shortest path and navigation (e.g. Dijkstra's algorithm) and compression (Huffman coding). In time based scheduling systems, time values, such as customer arrival time or, expected end of service time, are transformed into priorities that are then used in the PQ [2,3].

As noted first by Kleinrock, packet scheduling schemes are at the foundations of the successful construction of computer networks [4,2,5]. In today's routers and switches, PQs play a critical role in scheduling and deciding the transmission order of packets [6–8]. Priority Queues are

☆ A preliminary version of this paper appeared in the proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures, (SPAA 2013) [1].

* Corresponding author.
E-mail addresses: afek@post.tau.ac.il (Y. Afek), bremler@idc.ac.il (A. Bremler-Barr), schiffli@post.tau.ac.il (L. Schiff).

[1] Supported by the Israel Science Foundation Grant no. 1386/11.
[2] Supported by European Research Council (ERC) Starting Grant No. 259085.

http://dx.doi.org/10.1016/j.comnet.2014.03.010
1389-1286/

used to enforce fairness while also considering the different priorities of flows, thus guaranteeing that flows get a weighted (by their relative importance and history of usage) fair share of the bandwidth independent of the size of packets used.

Since PQs share the same time bounds as sorting algorithms [9], in high throughput scenarios, (e.g., backbone routers) special hardware PQs are used. Hardware PQs are usually implemented by ASIC chips that are specially tailored and optimized to the scenario and do not scale well [10–15].

We present a new construction for large hardware PQs, called Power Priority Queue (PPQ), which recursively uses small hardware priority queues in parallel as building blocks to construct a much larger one. The size of the resulting PQ is a power of the smaller PQs size, specifically we show that an n elements priority queue can be constructed from only $2k-1$ copies of any base (hardware) $\sqrt[k]{n}$ elements (size) priority queue. Our construction benefits from the optimized performance of small hardware PQs and extends these benefits to high performance, large size PQ.

We demonstrate the applicability of our construction in the case of the Ternary Content Addressable Memory (TCAM) based PQ, that was implied by Panigrahy and Sharma [16]. The TCAM based PQ, as we investigate and optimize in E, has poor scalability and becomes impractical when it is required to hold 1 M items. But by applying our construction with relatively tiny TCAM based PQ, we achieve a PQ of size 1 M with throughput of more than 100 M operations per second, which can be used to schedule packets at a line rate of 100 Gbps. The construction uses in parallel 10 TCAMs (or TCAM blocks) of size 110 Kb and each PQ operation requires 3.5 sequential TCAM accesses (3 for Dequeue and 4 for Insert).

Finally this work also improves the space and time performance of the TCAM based sorting scheme presented in [16]. As we show in Section 4 an n elements sorting algorithm is constructed from two $w\sqrt{n}$ entries TCAM's, where w is the number of bits required to represent one element (in [16] two n entries TCAM's are used). The time complexity to sort n elements in our solution is the same as in [16], $O(n)$, when counting TCAM accesses, however our algorithm accesses much smaller TCAM's and thus is expected to be faster. Moreover, in Section 4.2 we prove a lower bound on the time complexity of sorting n elements with a TCAM of size n (or $\sqrt{n}$) that matches our TCAM based sorting algorithm.

2. Priority queues background

2.1. Priority queues and routing

Since the beginning of computer networks, designing packet scheduling schemes has been one of the main difficulties [5,2]. In today's routers and switches, PQs play a critical role in scheduling and deciding the order by which packets are forwarded [6–8].

Priority Queues is the main tool with which the schedulers implement and enforce fairness combined with priority among the different flows.

Guaranteeing that flows get a weighted (by their relative importance) fair share of the bandwidth independent of packet sizes they use.

For example, in the popular Weighted Fair Queueing (WFQ) scheduler, each flow is given a different queue, ensuring that one flow does not overrun another. Then, different weights are associated with the different flows indicating their levels of quality of service and bandwidth allocation. These weights are then used by the WFQ scheduler to assign a time-stamp to each arriving packet indicating its virtual finish time according to emulated Generalized Processor Sharing (GPS). And now comes the critical and challenging task of the priority queue, to transmit the packets in the order of the lowest timestamp packet first, i.e., according to their assigned timestamps.[3] For example, in a 100 Gbps line rate, hundreds of thousands of concurrent flows are expected.[4] Thus the priority queue is required to concurrently hold more than million items and to support more than 100 million insert or dequeue operations per second. Note that the range of the timestamps depends on the router's buffer size and the accuracy of the scheduling system. For best accuracy, the timestamps should at least represent any offset in the router's buffer. Buffer size is usually set proportional to $RTT \cdot lineRate$, and for a 100 Gbps line rate and RTT of 250 ms, timestamp size can get as high as 35 bits.

No satisfactory software PQ implementation exists due to the inherent $O(\log n)$ step complexity per operation in linear space solutions, or alternatively $O(w)$ complexity but then with $O(2^w)$ space requirement, where n is the number of keys (packets) in the queue and w is the size of the keys (i.e., timestamps in the example above). These implementations are mostly based on binary heaps or Van De Boas Trees [12]. None of these solutions is scalable, nor can it handle large priority queues with reasonable performances.

Networking equipment designers have therefore turned to two alternatives in the construction of efficient high rate and high volume PQ's, either to implement approximate solutions, or to build complex hardware priority queues. The approximation approach has light implementation and does not require a PQ [18]. However the inaccuracy of the scheduler hampers its fairness, and is thus not applicable in many scenarios. The hardware approaches, described in detail in the next subsection, are on the other hand not scalable.

2.2. Hardware priority queue implementations

Here we briefly review three hardware PQ implementations, Pipelined heaps [13,19], Systolic Arrays [10,11] and Shift Registers [15]. ASIC implementations, based on pipelined heaps, can reach $O(1)$ amortized time per operation and $O(2^w)$ space [13,19], using pipeline depth that depends on w, the key size, or $\log n$ the number of elements. Due to the strong dependence on hardware design and key size, most of the ASIC implementations use small key size, and

[3] Note that it's enough to store the timestamp of the first packet per flow.

[4] Estimated by extrapolating the results in [17] to the current common rate.