A sketch data-structure for quantiles in BPF

Kornilios Kourtis
<kornilios@isovalent.com>

Outline

- Motivation
- Q-digest: a sketch data structure for quantiles
- Take a step back

Overall goal:

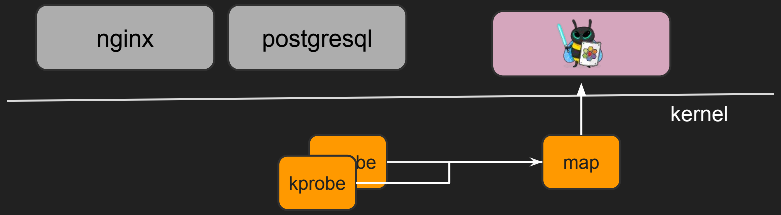
- Present a use-case, some first ideas, and solicit feedback

Workflow

- Observe events happening with BPF
- Provide statistical summaries to users
- Act based on summary values (e.g., to detect anomalies)

Maintaining summaries in kernel-side, allow us to:

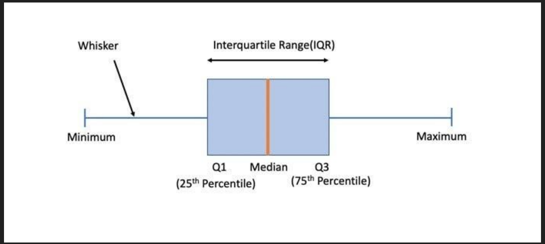
- reduce copying overhead
- act inline



Summaries

- means, standard deviations, min/max values
  - easy to incrementally maintain

- quantiles (e.g., 50%, 99%)
  - more interesting
  - hard to incrementally maintain: sort all values and pick the one you want
  - Sketches -> approximate results at abounded size (tradeoff)
    - Other examples: bloom filters / count-min / etc.



Q-digest

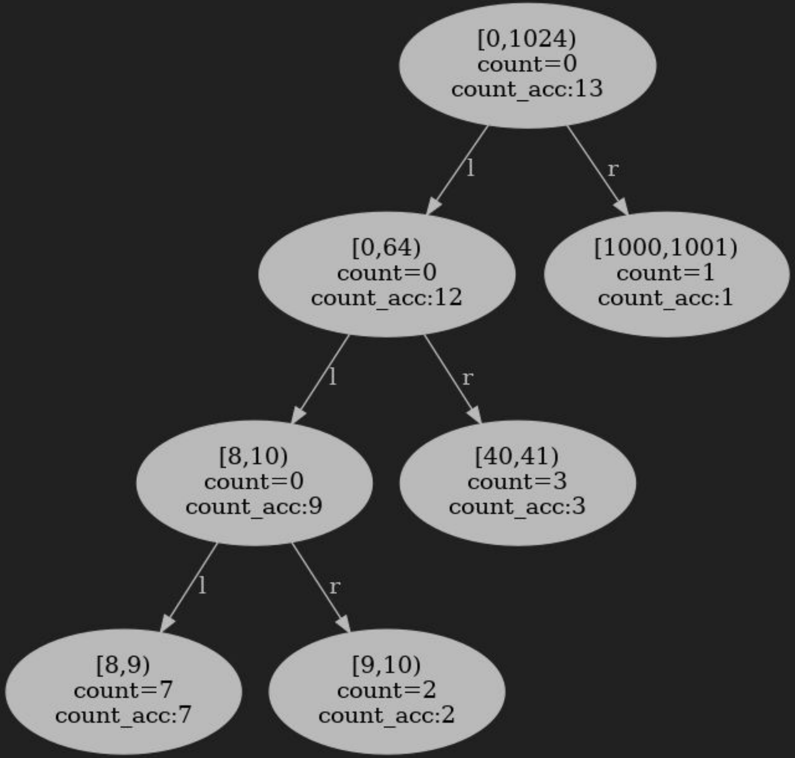
- “Medians and Beyond: New Aggregation Techniques for Sensor Networks”, Shrivastava et al, 2004
  - Describes the basic idea
  - Provides bounds on approximation error and size

Essentially:

- A binary trie
- Maintaining counts on $[x*2^d, (x+1)*2^d)$ ranges, where d is the level

- Pick this one and see how far we can get

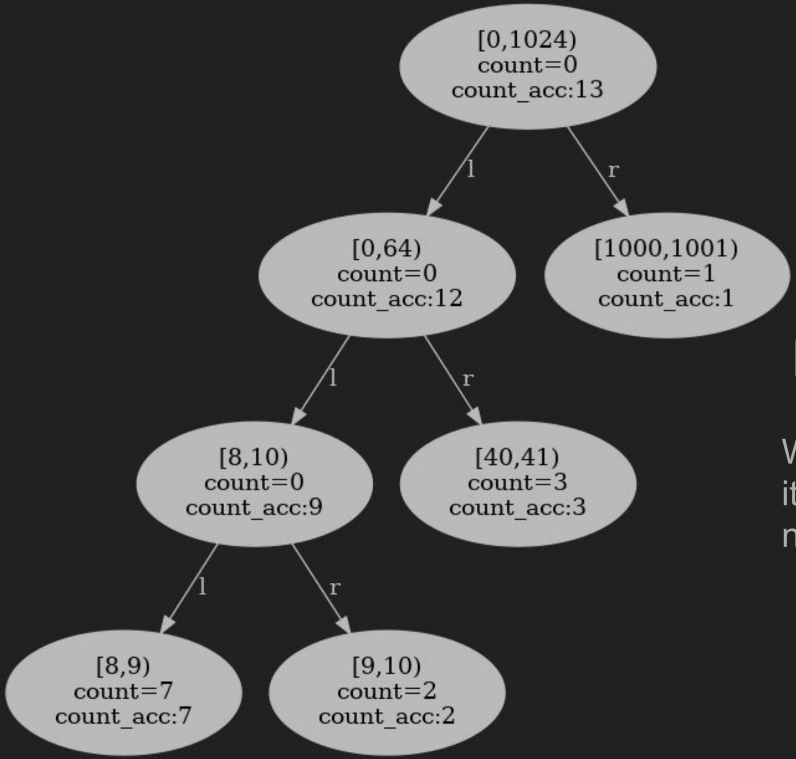
Q-digest: nodes



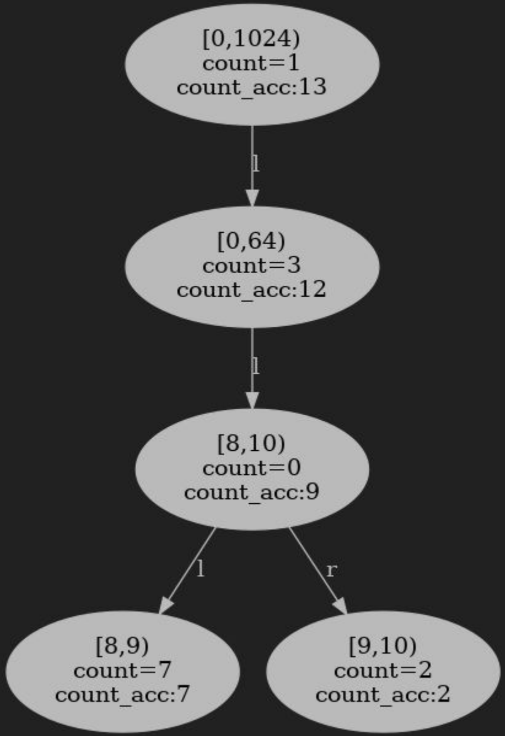
```
/* node represents range [a,b), where
 * a=(prefix+0)*(1<<suffix_len)
 * b=(prefix+1)*(1<<suffix_len) */
struct qdigest_node {
    struct rcu_head rcu;
    struct qdigest_node __rcu *child[2];
    __u64 count_acc;
    __u32 suffix_len;
    __u32 prefix;
};

struct qdigest_tree {
    struct bpf_map map;
    struct qdigest_node __rcu *root;
    spinlock_t lock;
};
```

Q-digest: compression

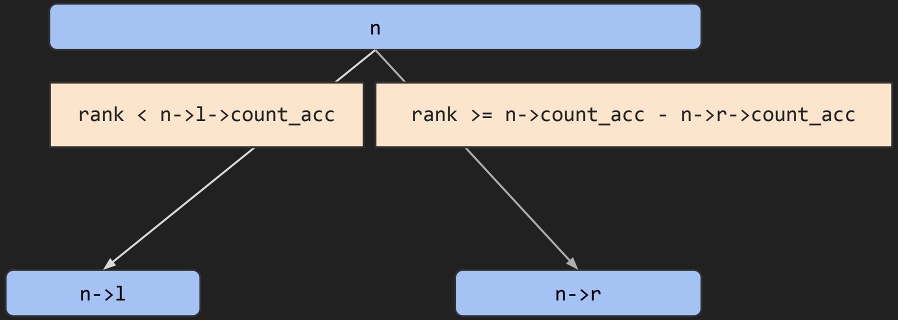


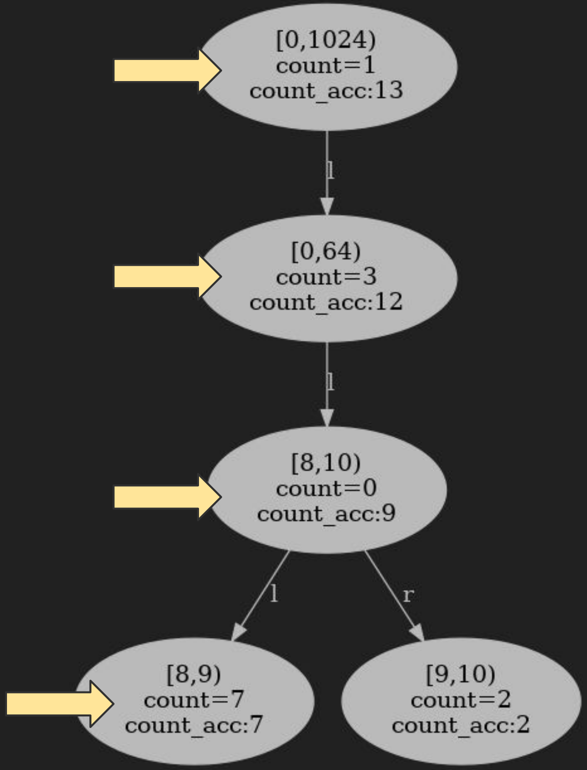
When `max_entries` nodes, iterate the tree and compress nodes



Q-digest: quantile lookup

- percentile query → rank query
  - E.g., 50% in 13 elements →rank 7
- Traverse the tree until the proper node is found
  - return its range (known error)





[0,1024) count=1 count_acc:13
l
[0,64) count=3 count_acc:12
l
[8,10) count=0 count_acc:9
l
r
[8,9) count=7 count_acc:7
[9,10) count=2 count_acc:2

A new BPF map?

- Original idea behind this talk was to propose a new bpf map

But:

- Q-digest is not a fundamental map like arrays or hash-tables
- There are many different data-structures implementing sketches (quantiles or other)
- Better to implement these data-structures in BPF

Q-digest using existing maps

- hash-table where key is <prefix>
- `bpf_loop()` for iteration
- bpf_spin_lock for while data structure

- Traversing the tree requires lookups instead of pointer dereferences

Can “modern” bpf implement Q-digest?

Implement it directly using “modern bpf”

- Owning vs non-owning references for new-style “graph” data-structures (linked_list, rbtree)
- Open-coded iterator loops (`bpf_iter_num`)
    - allocate and deallocate nodes

A map implemented in eBPF

- Let’s assume that we have bpf program that can implement the operations of Q-digest
- We can use the bpf code to access it from BPF
- How do we access it from user-space?
  - BPF_PROG_RUN (?)

- A bpf map based on bpf programs that

A map implemented in BPF

- We already pass fds on linking (`attr.link_create.target_fd`)
- `bpf_struct_ops`

Thanks!