Supplementary Materials of CuckooDuo

1 Pseudocode

Algorithm 1: Insertion workflow of CuckooDuo

Input: An item (key-value pair) $e = \langle key, value \rangle$

if $\mathcal{I}_1[h_1(key)]$ *or* $\mathcal{I}_2[h_2(key)]$ *has empty slot* **then**
    Insert $FP(key)$ into the empty slot in CuckooIndex;
    Insert e into the corresponding slot in CuckooVault;
    return;
Q := An empty queue used for BFS;
for *each fingerprint* f *in* $\mathcal{I}_1[h_1(key)]$ *and* $\mathcal{I}_2[h_2(key)]$ **do**
    Push $\langle f, path = f \rangle$ into the tail of Q;
while Q.`not_empty` **do**
    Pop $\langle f, path \rangle$ from the front of Q;
    if *length of path exceeds* L **then**
        return $failure$;
    if f *is stored in* $\mathcal{I}_1[h_1(f)]$ **then**
        $h_2(f) = (h_1(f) + hash(f))\%m$;
        if $\mathcal{I}_2[h_2(f)]$ *has empty slot* **then**
            Read all items on $path$ from CuckooVault;
            Write items on $path$ (including e) into CuckooVault;
            Write FPs on $path$ (including $FP(key)$) into CuckooIndex;
            return $success$;
        else if $\mathcal{I}_2[h_2(f)]$ *is full* **then**
            for *each fingerprint* f' *in* $\mathcal{I}_2[h_2(f)]$ **do**
                $path' \leftarrow (path, f')$;
                Push $\langle f', path' \rangle$ into the tail of Q;
    else if f *is stored in* $\mathcal{I}_2[h_2(f)]$ **then**
        $h_1(f) = (h_2(f) - hash(f))\%m$;
        Check $\mathcal{I}_1[h_1(f)]$ and perform similar operations as above;
return $failure$;

2 Mathematical Proofs

2.1 Proof for Theorem IV.1

Theorem IV.1. *Consider a basic CuckooDuo under the load factor of α. Let X_α be the number of items failed to be inserted into CuckooDuo due to fingerprint collisions. We have*

$$\mathbb{E}(X_\alpha) \approx 2md^2\alpha^2/2^f \leqslant 4md^2\alpha^2/2^f = O\left(\frac{md^2\alpha^2}{2^f}\right)$$

where m is the number of buckets in each bucket array, d is the bucket size, and f is the length of fingerprints (in bits).

Proof. Consider an incoming item e. The probability that its fingerprint collides with the fingerprint of another item is $1/2^f$. For a CuckooDuo under the load factor of α', the expected number of items in the two candidate buckets of e is $2d\alpha'$. Therefore, the expected probability of item e experiencing a fingerprint collision is

$$\mathcal{P} = 1 - (1 - 1/2^f)^{2d\alpha'} \approx 2d\alpha'/2^f$$

.

For the CuckooDuo under the load factor of α, we can calculate the expectation of X_α by integrating $\mathcal{P}$ over the number of inserted items x as

$$\mathbb{E}(X_\alpha) = \int_0^{2dm\alpha} \alpha' \cdot \frac{2d}{2^f} dx = \int_0^{2dm\alpha} \frac{x}{2dm} \cdot \frac{2d}{2^f} dx = \frac{2md^2\alpha^2}{2^f}$$

By fixing the load factor $\alpha' = \alpha$ during integration, we can also derive an upper bound as

$$\mathbb{E}(X_\alpha) \leqslant 4md^2\alpha^2/2^f$$

. □

2.2 Proof for Theorem IV.2

Theorem IV.2. *Consider a CuckooDuo with Dual-Fingerprint optimization. Let X be the number of items failed to be inserted into CuckooDuo due to fingerprint collisions. We have*

$$\mathbb{E}(X) \leqslant \frac{4md(d+1)(d-1)}{3 \cdot 2^{2f}} = O\left(\frac{md^3}{2^{2f}}\right)$$

where m is the number of buckets in each bucket array, d is the bucket size, and f is the length of fingerprints (in bits).

Proof. Let d_1 and d_2 be number of slots in each bucket of $\mathcal{I}_1$ using $FP_1(\cdot)$ and $FP_2(\cdot)$ respectively ($d = d_1 + d_2$). We assume $d_1 \geqslant 2$ and $d_2 \geqslant 2$.

Consider a certain bucket $\mathcal{I}_1[i]$ in the first bucket array. If the items in $\mathcal{I}_1[i]$ have only one fingerprint collision (either in FP_1 or FP_2), the collision can definitely be resolved through our *Dual-Fingerprint* adjustment. In the following, we assume that fingerprint collision occurs uniformly across all buckets. This assumption will lead to an upper bound of the failure probability because our *Dual-Fingerprint* algorithm can effectively resolve many collisions.

Let $\mathcal{P}$ be the probability that there exists fingerprint collisions in $\mathcal{I}_1[i]$ that cannot be resolved through *Dual-Fingerprint* adjustment. The upper bound of $\mathcal{P}$ can be written as $\overline{\mathcal{P}} = 1 - \mathcal{P}_0 - \mathcal{P}_1$, where $\mathcal{P}_j$ is the probability that j fingerprint collisions happen in the certain bucket $\mathcal{I}_1[i]$ ($j = 0$ means no fingerprint collision happens).

We derive $\mathcal{P}_0$ and $\mathcal{P}_1$ as

$$\mathcal{P}_0 = \left(\prod_{j=0}^{d-1}\left(1 - \frac{j}{2^f}\right)\right)^2$$

,

$$\mathcal{P}_1 = 2\left(\prod_{j=0}^{d-1}\left(1 - \frac{j}{2^f}\right)\right) \cdot \binom{d}{2} \cdot \frac{1}{2^f} \cdot \left(\prod_{j=0}^{d-2}\left(1 - \frac{j}{2^f}\right)\right)$$

where $\binom{d}{2} = \frac{d!}{2!(d-2)!}$ is the combination number.

Without loss of generality, we assume that $d < d^2 < d^3 \ll 2^f$. By using mathematical analysis techniques, we can get:

$$\mathcal{P}_0 = 1 - \frac{d(d-1)}{2^f} + \frac{d^2(d-1)^2 - 2/3 \cdot d(d-1)(2d-1)}{2^{2f}} + o\left(\frac{d^3}{2^{2f}}\right)$$

$$\mathcal{P}_1 = \frac{d(d-1)}{2^f} - \frac{d(d-1)^3}{2^{2f}} + o\left(\frac{d^3}{2^{2f}}\right)$$

$$\overline{\mathcal{P}} = 1 - \mathcal{P}_0 - \mathcal{P}_1 = \frac{d(d-1)(d+1)}{3 \cdot 2^{2f}} + o\left(\frac{d^3}{2^{2f}}\right)$$

Recall that we assume $d^3 \ll 2^f$, meaning that $\overline{\mathcal{P}}$ is very small. Therefore, for each item in bucket $\mathcal{I}_1[i]$, the probability that it has a fingerprint collision with another item in $\mathcal{I}_1[i]$ and the collision cannot be resolved with *Dual-Fingerprint* adjustment is $\mathcal{P}/d$.

As each item might collide with all items in its two candidate bucket, the upper bound of the probability that an item experiencing an unresolvable fingerprint collision is $2\overline{\mathcal{P}}/d$. Finally, as there are at most $2dm$ items in CuckooDuo, the expectation of the number of items with unresolvable collisions satisfies that

$$\mathbb{E}(X) \leqslant 2dm \cdot \frac{2\overline{\mathcal{P}}}{d} \leqslant 2dm \cdot \frac{2(d+1)(d-1)}{3 \cdot 2^{2f}} = \frac{4md(d+1)(d-1)}{3 \cdot 2^{2f}}$$

□

2.3 Proof for Theorem IV.3

Theorem IV.3. *Consider a basic CuckooDuo under the load factor of α. Let Y_α be the number of items failed to be inserted into CuckooDuo due to BFS failure (i.e., the length of kick-out path exceeds the predefined threshold L). We have*

$$\mathbb{E}\left(Y_\alpha\right) \approx 2md \int_0^\alpha \frac{\beta(r)^{2\Sigma_{i=0}^{L} d^i}}{1-\beta(r)^{2\Sigma_{i=0}^{L} d^i}} dr$$

where m is the number of buckets in each bucket array, d is the bucket size, L is the maximum length of kick-out path, and $\beta(r)$ is the ratio of full buckets under the load factor of r.

Proof. Consider a CuckooDuo under the load factor of r. We define its full bucket ratio as $\beta(r) := \frac{t(r)}{2m}$, where $t(r)$ is the number of full buckets in CuckooDuo. Here, $\beta(r)$ and $t(r)$ are two functions of load factor r.

For an incoming new item e, the probability that it cannot be directly inserted into one of its two candidate buckets is $\mathcal{P}_0 = \beta(r)^2$.

During a BFS process with the maximum kick-out path length of L, there are $2\Sigma_{i=0}^{L} d^i$ buckets to be checked. We assume the BFS process randomly visits each bucket. The probability that item e cannot be inserted into CuckooDuo through the BFS process is

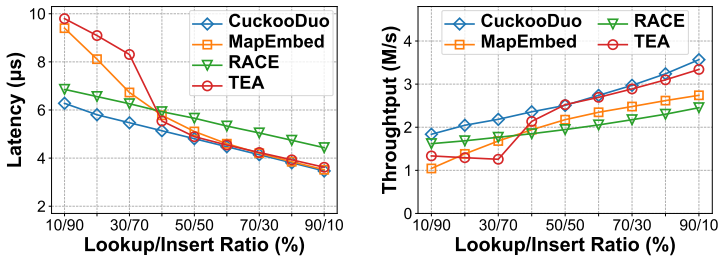
$$\mathcal{P}_L = \beta(r)^{2\Sigma_{i=0}^{L} d^i}$$

which is also the probability that the BFS cannot find a non-full bucket.

For the CuckooDuo under the load factor of α, we can calculate the expectation of Y_α by integrating over load factor r.

$$\mathbb{E}\left(Y_\alpha\right) \approx 2md \int_0^\alpha \frac{\mathcal{P}_L}{1-\mathcal{P}_L} dr = 2md \int_0^\alpha \frac{\beta(r)^{2\Sigma_{i=0}^{L} d^i}}{1-\beta(r)^{2\Sigma_{i=0}^{L} d^i}} dr$$

□



(a) Latency (b) Throughput

Figure 1: Comparison of speed on hybrid workloads.

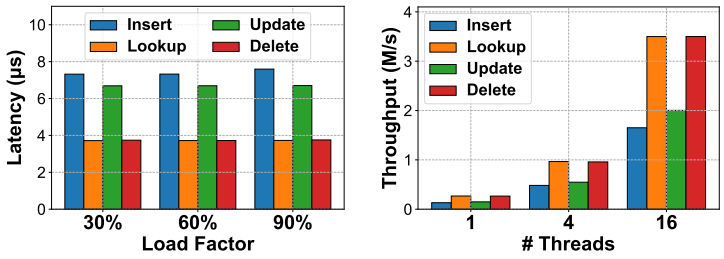
3 Additional Experimental Results

3.1 Performance on Hybrid Workloads

Comparison of average speed on hybrid workloads (Figure 1): We find that CuckooDuo performs the best under hybrid workloads. We create nine YCSB hybrid workloads with different lookup/insert ratios. We first load 3M items into the four algorithms to attain a 10% load factor, and then run the nine hybrid workloads to measure the average latency and throughput. For the throughput experiments, we use multi-threading acceleration with 16 threads. The results show that with the increase of lookup/insert ratio, the latency of all algorithms decreases and the throughput increases. Due to having both high lookup and insert performance, CuckooDuo always has the best latency and throughput, which is at most $1.6\times$ and $1.9\times$ better than prior art.

3.2 Performance on Large-scale Workloads

We evaluate CuckooDuo on large-scale workloads. We create a YCSB workload of 1G distinct insert requests, and create three workloads of 1G lookup/update/delete requests (following default Zipfian distribution $\theta = 0.99$). The table size (# slots) of CuckooDuo is set to 1G. We find that on large-scale workloads, CuckooDuo still has small latency and high throughput. As shown in Figure 2(a), the insert/lookup/update/delete latency is about 7.32/3.71/6.68/3.72 μs, which is $0.4 \sim 0.8 \mu s$ larger than that in small-scale experiments. As shown in Figure 2(b), when using 16 threads, the insert/lookup/update/delete throughput is $1.65/3.50/2.01/3.50$ M/s, which is $0 \sim 0.1$ M/s smaller than that in small-scale experiments. On large-scale experiments, the speed of CuckooDuo slightly drops because of the increased time taken by the BFS procedure in local server, which is incurred by cache factors.



(a) Latency (b) Throughput

Figure 2: Performance on large-scale workloads.