2025 AdvDS HW4

B12902036 吳述宇

May 9, 2025

Introduction

In this assignment, we implement and compare three classical streaming algorithms—Morris++, FM++, and HyperLogLog, which estimate counts in $o(\log N)$ space, where N denotes the total number of stream elements. Morris++ provides an estimate of the total stream length, while FM++ and HyperLogLog focus on estimating the number of distinct elements, leveraging hash-based sketching techniques. Among them, FM++ serves as a well-studied baseline, and HyperLogLog is widely adopted in practice for its performance on accuracy and memory usage.

To evaluate their performance, we generate three payloads of size $N = 10^5$—(1) all-new elements, (2) all-identical elements, and (3) random duplicates, and repeat each trial 30 times. We measure mean relative error, 99th-percentile error, and memory consumption to assess accuracy, stability, and efficiency. Finally, a provided Python script visualizes these metrics across all experiments.

Algorithm Overview

In this section, we briefly describe the three implemented streaming sketches and the chosen parameter settings.

1. Morris++ : Estimate total stream length with performing randomized increments with probability 2^{-c} on multiple counters and apply the median trick to reduce variance. It uses $O(\ell \cdot k)$ counters [Mor78].

2. FM++ : Estimate number of distinct elements with hashing each element, counting leading zeros in the hashed value, averaging across k counters and take median of ℓ groups [DF03]. It uses $O(\ell \cdot k)$ counters.

3. HyperLogLog : Estimate number of distinct elements with $m = 2^b$ registers to record the maximum leading-zero streak per bucket, then apply harmonic-mean estimation with bias-correction [FFGM07].

4. **Parameter Settings** We set $k = 100$, $\ell = 6$, $b = 7$ (so $m = 128$), stream length $N = 10^5$, and 30 trials. These choices balance variance reduction against space consumption and ensure meaningful comparison under similar memory budgets.

Evaluation Methodology

We test each algorithm on three stream types of length $N = 10^5$, generated under a fixed random seed:

- **UNIQUE_RANDOM:** Permutation of $[0, 10^5)$ shuffled with `std::shuffle`.
- **ALL_SAME:** The constant value repeated 10^5 times.
- **MANY_DUPLICATES:** Random samples from $\{0, \ldots, 999\}$.

We perform $T = 30$ independent trials per stream and report:

- **Mean Relative Error :** $\frac{1}{T}\sum_{t=1}^{T}\frac{|\hat{X}_t - X|}{X}$.
- **99th Percentile Error:** the 99th-percentile of the 30 Relative Error samples.
- **Estimated Memory Usage**

Results and Analysis

In this section, we present the experimental results and analyze the performance of the three algorithms under different types of input payloads. Each configuration is tested with 30 independent trials, and the evaluation metrics include average relative error, 99th percentile error, and sketch memory usage.

1. Accuracy across payloads. For `UNIQUE_RANDOM` and `MANY_DUPLICATES` in 1, HLL consistently achieves the lowest relative error, while FM++ exhibits moderate overestimation, and Morris++ shows small bias. In the `ALL_SAME` case, FM++ completely fails due to its reliance on hash diversity, while HLL remains accurate.

2. Robustness. The relative errors of HLL remain almost constant across all trials due to the deterministic hash function and fixed input; this explains why its P99 errors, which is in 3, are identical to its average. In contrast, FM++ and Morris++ show visible fluctuations, especially FM++, which is more sensitive to the hash collision pattern.

3. Space efficiency. As shown in Figure 2, HLL uses significantly less memory compared to Morris++ and FM++. This space advantage is notable given HLL's superior accuracy.

4. Trade-offs. While FM++ is classic, its sensitivity to hash collisions limits its reliability in low-entropy input cases. Morris++ is more robust but does not target distinct counting. HLL offers the best practical performance overall, both in accuracy and space usage.

Conclusion

In this work, we have implemented and compared three classical streaming sketches, each operating in $o(\log N)$ space, across three representative data streams of length $N = 10^5$. Our key findings are as follows:

- **HyperLogLog** achieves the best overall performance, with the lowest mean relative error and 99th-percentile error on both the UNIQUE_RANDOM and MANY_DUPLICATES payloads. Even in the ALL_SAME case, HLL maintains a low error, using only 128 registers.
- **FM++** performs acceptably on high-entropy inputs, but degrades substantially when hash diversity is limited. Its error climbs in MANY_DUPLICATES and ALL_SAME, highlighting its fragility under low entropy or adversarial patterns.
- **Morris++** reliably estimates total stream length with moderate mean relative error and 99th-percentile error, but it uses the most space among the three.

Beyond these results, our study underscores several practical considerations often abstracted in theoretical analyses:

1. **Hash Function Selection** : The choice of a high-quality, uniform hash function proved critical. In my experience of implementation, poor hash mixing can inflate FM++'s variance and bias HLL' registers.
2. **Entropy Sensitivity** : Sketches that rely on randomization must handle low-entropy streams; otherwise, worst-case error can far exceed expectations.

Also, we propose several opinions for future research and engineering:

- **Adaptive Precision** : Develop sketches that dynamically adjust their parameter b (in HLL) or group size k, ℓ based on online estimates of stream entropy or observed collision rates, thereby optimizing space-accuracy trade-offs in real time. If the researchers can figure out the best parameters for certain usage, the engineer's time for tuning can be saved.
- **Hybrid Sketch Architectures** : Combine multiple sketch types, for example, pairing a Count-Min sketch for heavy hitters with a HyperLogLog for cardinality to achieve more robust estimates under mixed workloads. However, the engineering process needs creativity and can be difficult.
- **Reproducibility and Benchmarking** : Standardize benchmarking protocols, including fixed RNG seeds, common payload generators, and shared datasets to facilitate fair comparisons and drive adoption of best-in-class sketches for both academia and industry respectively.

In addition to HyperLogLog, which delivered the best performance in our experiments, several refinements have been proposed over time. For example, the Compressed Probabilistic Counting (CPC) sketch in [Lan17] achieves the same accuracy as traditional HyperLogLog while reducing space by roughly 20–40%. However, this improvement comes at the cost of significantly more complex implementation, particularly due to low-level bit-packing logic and increased maintenance overhead.

Ultimately, this assignment underscores the importance of bridging the gap between theoretical guarantees and practical deployment because it demands not only rigorous mathematical analysis but also meticulous attention to implementation details.

Appendix

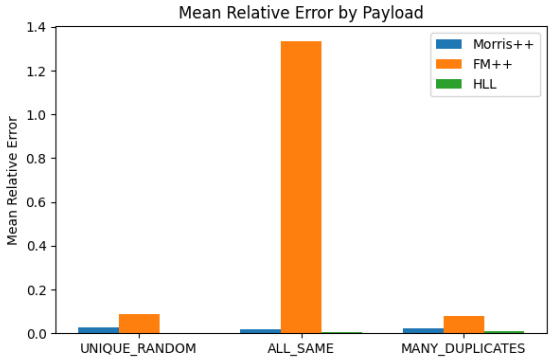

Figure 1: Mean Relative Error

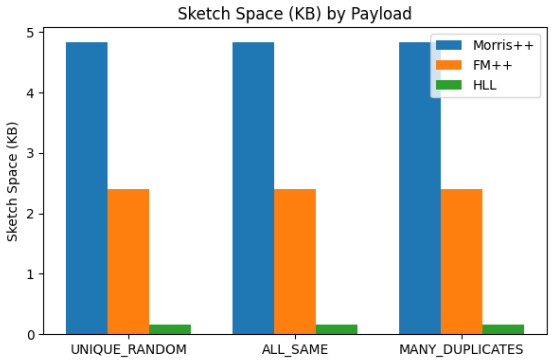

Figure 2: Space

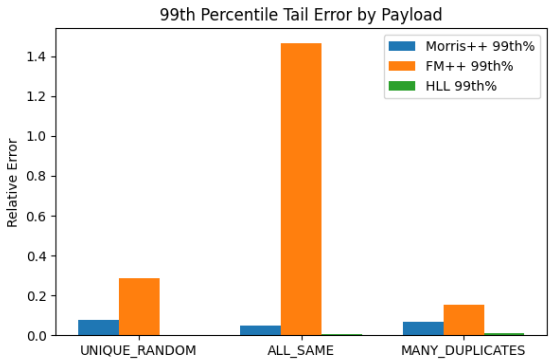

Figure 3: Tail Error

References

[DF03] Matthieu Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *ESA*, pages 605–617. Springer, 2003.

[FFGM07] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *AOFA*, pages 127–146, 2007.

[Lan17] Kevin J. Lang. Back to the future: an even more nearly optimal cardinality estimation algorithm. https://arxiv.org/abs/1708.06839v1, August 2017. Oath Research, August 22, 2017.

[Mor78] Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.