



Đề cương luận văn thạc sĩ

Nghiên cứu phát triển kỹ thuật đếm số phần tử
trên dòng dữ liệu

Học viên: Lê Anh Quốc

ID: 2070428

Người hướng dẫn khoa học:

PGS. TS. THOẠI NAM

1. Giới thiệu
2. Các công trình nghiên cứu liên quan
3. Phát biểu bài toán
4. Mục tiêu, đối tượng và giới hạn nghiên cứu
5. Cơ sở lý thuyết
6. Phương pháp thực hiện
7. Kế hoạch triển khai

Giới thiệu

- Ứng dụng và dịch vụ trực tuyến đóng vai trò quan trọng trong cuộc sống hiện đại.
- DAU là chỉ số quan trọng để đánh giá hiệu quả hoạt động của các ứng dụng và dịch vụ này.
- Theo dõi DAU giúp:
 - Đánh giá mức độ tương tác và quan tâm của người dùng.
 - Đo lường hiệu quả của chiến dịch marketing và quảng cáo.
 - Hỗ trợ ra quyết định kinh doanh.

- Thách thức:
 - Đếm DAU trên dữ liệu lớn và tốc độ truy cập cao.
 - Tổng hợp DAU từ nhiều nguồn dữ liệu khác nhau.
 - Đếm DAU theo nhiều khoảng thời gian và tiêu chí khác nhau.
- Giải pháp:
 - Sử dụng các thuật toán đếm hiệu suất cao và tin cậy.
 - Xây dựng hệ thống tổng hợp dữ liệu linh hoạt và đồng bộ.
 - Áp dụng các kỹ thuật phân tích dữ liệu chi tiết và chuyên sâu.

Các công trình nghiên cứu liên quan

Các công trình nghiên cứu liên quan

- Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier
LogLog Counting of Large Cardinalities, 2003 [1]:
 - Thuật toán ước lượng số lượng phần tử với độ chính xác cao và sử dụng ít bộ nhớ.
- Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier
HyperLogLog: The Analysis of a Near-Optimal Cardinality Estimation Algorithm, 2007 [2]:
 - là một cải tiến từ LogLog, thuật toán này có khả năng ước lượng số lượng phần tử lớn hơn 10^9 với độ chính xác khoảng 2% chỉ dùng 1.5 kilobytes bộ nhớ, đồng thời có khả năng song song hoá tối ưu và thích nghi với mô hình cửa sổ trượt (sliding window).
- Stefan Heule, Marc Nunkesser, Alexander Hall
HyperLogLog in Practice: Algorithmic Improvements for Practical Cardinality Estimation Deployments, 2017 [3]:
 - Phiên bản nâng cấp của HyperLogLog với độ chính xác cao hơn và yêu cầu bộ nhớ ít hơn.

Các công trình nghiên cứu liên quan

- Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier, LogLog Counting of Large Cardinalities, 2010 [4]:
 - Cung cấp ước lượng chính xác số lượng phần tử duy nhất trong dòng dữ liệu liên tục theo cơ chế cửa sổ trượt, sử dụng bộ nhớ hiệu quả. Nó đặc biệt hữu ích cho giám sát thời gian thực và phân tích dữ liệu liên tục.
- Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier, HyperLogLog: The Analysis of a Near-Optimal Cardinality Estimation Algorithm, 2024 [5]:
 - là một cấu trúc dữ liệu mới cho việc đếm độc lập xấp xỉ, tương tự như HyperLogLog, nhưng tiêu tốn ít hơn 43% không gian với cùng lỗi ước lượng.

Phát biểu bài toán

- **Bài toán 1:** Phát triển thuật toán để ước lượng số lượng phần tử (cardinality estimation) trong một khoảng thời gian trên một dòng dữ liệu (data stream).
- **Bài toán 2:** Mở rộng thuật toán để ước lượng số lượng phần tử trong một khoảng thời gian trên nhiều dòng dữ liệu.

Mục tiêu, đối tượng và giới hạn nghiên cứu

Mục tiêu chính:

- Phát triển kỹ thuật đếm số lượng phần tử hiệu quả, có chính xác cao trên dòng dữ liệu.
- Nâng cao hiệu suất xử lý dữ liệu lớn, đáp ứng nhu cầu ngày càng tăng trong kỷ nguyên số.
- Đóng góp vào sự phát triển của công nghệ dữ liệu lớn, mở ra tiềm năng ứng dụng rộng lớn trong nhiều lĩnh vực.

Mục tiêu cụ thể:

- Phân tích và đánh giá các kỹ thuật đếm số lượng phần tử hiện có.
- Đề xuất và triển khai kỹ thuật đếm số lượng phần tử mới, tối ưu hóa hiệu suất và độ chính xác.
- Thực hiện thí nghiệm để chứng minh tính ưu việt của kỹ thuật mới so với các kỹ thuật hiện có.
- Phân tích kết quả thí nghiệm, rút ra kết luận và đề xuất hướng nghiên cứu tiếp theo.

- Đề tài tập trung nghiên cứu kỹ thuật đếm số lượng phần tử trên dòng dữ liệu dạng văn bản.
- Các kỹ thuật được đề xuất và triển khai có thể chưa áp dụng được cho tất cả các loại dữ liệu.
- Nghiên cứu chỉ giới hạn trong thời gian cho phép.

Đối tượng nghiên cứu

Đối tượng nghiên cứu của đề tài "Nghiên cứu phát triển kỹ thuật đếm số lượng phần tử trên dòng dữ liệu" Đối tượng nghiên cứu chính:

- Dòng dữ liệu dạng văn bản có chứa nhiều phần tử cần đếm.
- Các kỹ thuật đếm số lượng phần tử hiện có và mới được đề xuất.

Đối tượng nghiên cứu cụ thể:

- Số phần tử có thể là userID, IP address hoặc bất kỳ đối tượng nào tương đương mà có thể đếm được nhờ định danh của nó.
- Các ứng dụng xử lý dữ liệu lớn có nhu cầu đếm số lượng phần tử hiệu quả.

Cơ sở lý thuyết

- Các thuật toán xác suất cho việc ước lượng trong thực tế là họ của LogLog:
 - *LogLog*, 2003
 - *HyperLogLog*, 2007
 - *HyperLogLog++*, 2017
- Sử dụng phương pháp tương tự thuật toán Đếm Xác Suất
 - ước lượng số lượng n bằng cách quan sát số lượng lớn nhất của các số không dấu trong biểu diễn nhị phân
 - Hàm băm $h \leftarrow \{0, 1, \dots, 2^M - 1\}$, với độ dài M

$$h(x) = j = \sum_{k=0}^{M-1} j_k \cdot 2^k := (i_0 i_1 \dots i_{M-1})_2, i_k \in \{0, 1\}$$

- Bằng cách sử dụng một phần của giá trị băm $h(x)$ để chia tập dữ liệu ban đầu thành một số tập con.
- Phần còn lại được sử dụng để cập nhật bộ đếm dựa trên việc quan sát mẫu $0^k 1$
- Hạng tương đương với vị trí bit 1 đầu tiên từ trái qua phải (rank)

$$\text{rank}(i) = \begin{cases} \min_{i_k \neq 0}, & \text{for } i > 0, \\ M & \text{for } i = 0 \end{cases}$$

- Tiết kiệm dung lượng lưu trữ và có độ chính xác cao.

LogLog algorithm

Ý tưởng cơ bản của thuật toán *LogLog* bắt đầu bằng việc tính toán hạng cho mỗi phần tử đầu vào dựa trên một hàm băm đơn giản h . Vì chúng ta có thể mong đợi rằng khoảng $\frac{n}{2^k}$ phần tử có thể có $rank(\cdot) = k$, trong đó n là tổng số phần tử được lập chỉ mục vào một bộ đếm, hạng quan sát tối đa có thể cung cấp một dấu hiệu tốt về giá trị của $\log_2 n$:

$$R = \max_{x \in D} (rank(x)) \approx \log_2 n.$$

Tuy nhiên, ước lượng như vậy có sai số khoảng ± 1.87 lần nhị phân, điều này không thực tế. Để giảm sai số, thuật toán *LogLog* sử dụng một kỹ thuật phân nhóm dựa trên việc trung bình ngẫu nhiên và chia tập dữ liệu thành $m = 2^p$ tập con S_0, S_1, \dots, S_{m-1} , trong đó tham số độ chính xác p xác định số bit được sử dụng trong điều hướng.

Do đó, đối với mỗi phần tử x từ tập dữ liệu, p bit đầu tiên của giá trị băm $h(x)$ M-bit có thể được lấy để tìm ra chỉ số j của tập con thích hợp.

$$j = (i_0 i_1 \dots i_{p-1})_2,$$

và phần còn lại (M-p) bit được lập chỉ mục vào bộ đếm tương ứng

COUNTER[i] để tính hạng và quan sát R theo công thức (3.9)

LogLog algorithm

Algorithm 1: Estimating cardinality with *LogLog*

Input: Dataset D

Input: Array of m *LogLog* counters with hash function h

Output: Cardinality estimation

$COUNTER[j] \leftarrow 0, j = 0 \dots m - 1$

for $x \in D$ **do**

$i \leftarrow h(x) := (i_0 i_1 \dots i_{M-1})_2, i_k \in \{0, 1\}$

$j \leftarrow (i_0 i_1 \dots i_{M-1})_2$

$r \leftarrow \text{rank}((i_p i_{p+1} \dots i_{M-1})_2)$

$COUNTER[j] \leftarrow \max(COUNTER[j], r)$

end

$R \leftarrow \frac{1}{m} \sum_{k=0}^{m-1} COUNTER[j]$

return $\alpha_m \cdot m \cdot 2^R$

LogLog algorithm

Sai số tiêu chuẩn δ của thuật toán *LogLog*:

$$\delta \approx \frac{1.3}{\sqrt{m}}$$

- $m = 256$, $\delta \approx 8\%$
- $m = 1024$, nó giảm xuống còn khoảng 4%.

Yêu cầu lưu trữ của thuật toán *LogLog* có thể được ước tính là $O(\log_2 \log_2 n)$ bit nếu cần đếm đến n . Cụ thể hơn, tổng không gian được yêu cầu bởi thuật toán để đếm đến n là $m \cdot \log_2 \log_2 \frac{n}{m} (1 + O(1))$. So sánh với thuật toán Đếm Xác suất trong đó mỗi bộ đếm yêu cầu 16 hoặc 32 bit, thuật toán *LogLog* yêu cầu bộ đếm nhỏ hơn nhiều $\{COUNTER[j]\}_{j=0}^{m-1}$, thường là 5 bit mỗi bộ đếm. Tuy nhiên, trong khi thuật toán *LogLog* cung cấp hiệu quả lưu trữ tốt hơn so với thuật toán Đếm Xác suất, nó đôi chút ít chính xác hơn.

Giả sử chúng ta cần đếm định lượng cho đến 2^{30} , tức là khoảng 1 tỷ, với độ chính xác khoảng 4%. Như đã đề cập, cho sai số tiêu chuẩn như vậy, cần $m = 1024$ ngăn, mỗi ngăn sẽ nhận xấp xỉ $\frac{n}{m} = 2^{20}$ phần tử

HyperLogLog algorithm

Một cải tiến của thuật toán *LogLog*, gọi là *HyperLogLog*, đã được đề xuất bởi Philippe Flajolet, Eric Fusy, Olivier Gandouet và Frederic Meunier vào năm 2007 [2]. Thuật toán *HyperLogLog* sử dụng hàm băm 32-bit, một hàm đánh giá khác nhau và các sửa lỗi bias khác nhau.

Tương tự như thuật toán *LogLog*, *HyperLogLog* sử dụng ngẫu nhiên hóa để xấp xỉ định lượng của một tập dữ liệu và đã được thiết kế để xử lý các định lượng lên đến 10^9 với một hàm băm 32-bit đơn lẻ h chia tập dữ liệu thành $m = 2^p$ tập con, với độ chính xác $p \in 4 \dots 16$. Ngoài ra, hàm đánh giá khác biệt thuật toán *HyperLogLog* so với *LogLog* tiêu chuẩn. Thuật toán *LogLog* gốc sử dụng trung bình hình học (geometric mean) trong khi *HyperLogLog* sử dụng một hàm dựa trên phiên bản chuẩn hóa của trung bình điều hòa (harmonic mean):

$$\hat{n} \approx \alpha_m \cdot m^2 \cdot \left(\sum_{j=0}^{m-1} 2^{-\text{COUNTER}[j]} \right),$$

với

$$\alpha_m = \left(m \int_0^\infty \left(\log_2 \left(\frac{2+x}{2} \right) \right)^m dx \right)^{-1}.$$

HyperLogLog algorithm

Ý tưởng đằng sau việc sử dụng trung bình điều hòa là nó giảm phương sai do tính chất của nó để kiểm soát các phân phối xác suất lệch.

Table 1: α_m for most used values of m

m	α_m
64	0.673
256	0.697
1024	0.709
$\geq 2^7$	$\frac{0.7213 \cdot m}{m+1.079}$

HyperLogLog algorithm

Tuy nhiên, ước lượng \hat{n} yêu cầu một sự điều chỉnh cho các phạm vi nhỏ và lớn do lỗi phi tuyến. Flajolet và các đồng nghiệp đã tìm thấy từ kinh nghiệm rằng đối với các định lượng nhỏ $n < \frac{5}{2}m$ để đạt được ước lượng tốt hơn, thuật toán *HyperLogLog* có thể được sửa lỗi bằng Đếm Tuyến Tính bằng cách sử dụng một số bộ đếm $COUNTER[j]$ khác không (nếu một bộ đếm có giá trị là không, chúng ta có thể nói chắc chắn rằng tập con cụ thể đó là trống).

Do đó, cho các phạm vi định lượng khác nhau, được biểu diễn dưới dạng các khoảng trên ước lượng \hat{n} , thuật toán cung cấp các sửa lỗi sau:

$$n = \begin{cases} \text{LINEARCOUNTER}, & \hat{n} \leq \frac{5}{2}m \text{ and } \exists j : COUNTER[j] \neq 0 \\ -2^{32} \log \left(1 - \frac{\hat{n}}{2^{32}}\right), & \hat{n} > \frac{1}{30}2^{32} \\ \hat{n}, & \text{otherwise.} \end{cases}$$

HyperLogLog algorithm

Tuy nhiên, đối với $n = 0$, sự sửa lỗi có vẻ không đủ và thuật toán luôn trả về một kết quả xấp xỉ $0.7m$.

Vì thuật toán *HyperLogLog* sử dụng một hàm băm 32-bit, khi định lượng tiến gần đến $2^{32} \approx 4 \cdot 10^9$, hàm băm gần như đạt đến giới hạn của nó và xác suất va chạm tăng lên. Đối với các phạm vi lớn như vậy, thuật toán *HyperLogLog* ước lượng số lượng giá trị băm khác nhau và sử dụng nó để xấp xỉ định lượng. Tuy nhiên, trong thực tế, có nguy cơ rằng một số lượng cao hơn không thể được đại diện và sẽ bị mất, ảnh hưởng đến độ chính xác.

Xem xét một hàm băm mà ánh xạ vũ trụ thành các giá trị có độ dài M bit. Tối đa, một hàm như vậy có thể mã hóa 2^M giá trị khác nhau và nếu định lượng ước lượng n tiến dần đến giới hạn này, độ lệch hàm băm trở nên ngày càng có khả năng xảy ra.

Không có bằng chứng cho thấy một số hàm băm phổ biến (ví dụ: MurmurHash3, MD5, SHA-1, SHA-256) hoạt động đáng kể tốt hơn các hàm khác trong các thuật toán *HyperLogLog* hoặc các biến thể của nó.

HyperLogLog algorithm

Algorithm 2: Estimating cardinality with *HyperLogLog*

Input: Dataset D

Input: Array of m *LogLog* counters with hash function h

Output: Cardinality estimation

$COUNTER[j] \leftarrow 0, j = 0 \dots m - 1$

for $x \in D$ **do**

$i \leftarrow h(x) := (i_0 i_1 \dots i_{M-1})_2, i_k \in \{0, 1\}$

$j \leftarrow (i_0 i_1 \dots i_{M-1})_2$

$r \leftarrow \text{rank}((i_p i_{p+1} \dots i_{M-1})_2)$

$COUNTER[j] \leftarrow \max(COUNTER[j], r)$

end

$R \leftarrow \frac{1}{m} \sum_{k=0}^{m-1} COUNTER[j]$

$\hat{n} = \alpha_m \cdot m^2 \cdot \frac{1}{R}$

$n \leftarrow \hat{n}$

if $\hat{n} \leq \frac{5}{2}m$ **then**

$Z \leftarrow \text{count}_{j=0 \dots m-1} (COUNTER[j] == 0)$

if $Z < 0.5$ **then**

HyperLogLog algorithm

Tương tự như thuật toán *LogLog*, có một sự đánh đổi rõ ràng giữa sai số tiêu chuẩn δ và số lượng bộ đếm m :

$$\delta \approx \frac{1.04}{\sqrt{m}}.$$

Yêu cầu bộ nhớ không tăng tuyến tính theo số lượng phần tử (không giống như, ví dụ, thuật toán Đếm Tuyến Tính), phân bổ ($M = p$) bit cho các giá trị băm và có tổng cộng $m = 2^p$ bộ đếm, bộ nhớ cần thiết là

$$\lceil \log_2 (M + 1 - p) \rceil \cdot 2^p \text{ bits}$$

hơn nữa, vì thuật toán chỉ sử dụng hàm băm 32-bit và độ chính xác $p \in 4 \dots 16$, yêu cầu bộ nhớ cho cấu trúc dữ liệu *HyperLogLog* là $5 \cdot 2^p$ bit. Do đó, thuật toán *HyperLogLog* cho phép ước lượng các định lượng vượt xa 10^9 với độ chính xác thông thường là 2% trong khi chỉ sử dụng một bộ nhớ chỉ 1.5 KB.

Ví dụ, cơ sở dữ liệu trong bộ nhớ nổi tiếng Redis duy trì cấu trúc dữ liệu *HyperLogLog* của 12 KB để xấp xỉ các định lượng với sai số tiêu chuẩn là 0.81%. Trong khi *HyperLogLog*, so với *LogLog*, đã cải thiện ước lượng

HyperLogLog++ algorithm

Sau một thời gian, vào năm 2013 [4], một phiên bản cải tiến của *HyperLogLog* đã được phát triển, đó là thuật toán *HyperLogLog++*, được công bố bởi Stefan Heule, Marc Nunkesser và Alexander Hall và tập trung vào các định lượng lớn hơn và sửa lỗi sai tốt hơn.

Cải tiến đáng chú ý nhất của thuật toán *HyperLogLog++* là việc sử dụng hàm băm 64-bit. Rõ ràng, càng dài giá trị đầu ra của hàm băm, càng nhiều phần tử khác nhau có thể được mã hóa. Sự cải thiện này cho phép ước lượng các định lượng lớn hơn rất nhiều so với 10^9 phần tử duy nhất, nhưng khi định lượng tiến gần đến $2^{64} \approx 1.8 \cdot 10^{19}$, độ chính xác của hàm băm cũng trở thành một vấn đề đối với *HyperLogLog++*.

Thuật toán *HyperLogLog++* sử dụng chính xác hàm đánh giá giống như được đưa ra trong *HyperLogLog*. Tuy nhiên, nó cải thiện việc sửa lỗi sai. Các tác giả của thuật toán đã thực hiện một loạt các thí nghiệm để đo lường sai lệch và phát hiện rằng đối với $n \leq 5m$, sai lệch của thuật toán *HyperLogLog* gốc có thể được sửa lỗi hơn bằng cách sử dụng dữ liệu thực nghiệm được thu thập trong quá trình thí nghiệm.

Ngoài bài báo gốc, Heule và đồng nghiệp cung cấp các giá trị được xác

HyperLogLog++ algorithm

Algorithm 3: Correcting bias in *HyperLogLog++*

Input: Estimate \hat{n} with precision p

Output: Bias-corrected cardinality estimate

$n_{low} \leftarrow 0, n_{up} \leftarrow 0, j_{low} \leftarrow 0, j_{up} \leftarrow 0$

for $j \leftarrow 0$ to $\text{length}(\text{RAWESTIMATEDATA}[p-4])$ **do**

if $\text{RAWESTIMATEDATA}[p-4][j] \geq \hat{n}$ **then**

$j_{low} \leftarrow j - 1, j_{up} \leftarrow j$

$n_{low} \leftarrow \text{RAWESTIMATEDATA}[p-4][j_{low}]$

$n_{up} \leftarrow \text{RAWESTIMATEDATA}[p-4][j_{up}]$

break

end

end

$b_{low} \leftarrow \text{BIASDATA}[p-4][j_{low}]$

$b_{up} \leftarrow \text{BIASDATA}[p-4][j_{up}]$

$y = \text{interpolate}((n_{low}, n_{low} - b_{low}), (n_{up}, n_{up} - b_{up}))$

return $y(\hat{n})$

HyperLogLog++ algorithm

Ví dụ (1), giả sử chúng ta đã tính toán ước lượng định lượng $\hat{n} = 2018.34$ bằng cách sử dụng công thức

$$\hat{n} \approx \alpha_m \cdot m^2 \cdot \left(\sum_{j=0}^{m-1} 2^{-\text{COUNTER}[j]} \right),$$

và muốn sửa chúng cho độ chính xác $p = 10 (m = 2^{10})$.

Đầu tiên chúng ta kiểm tra mảng RAWESTIMATEDATA[6] và xác định giá trị \hat{n} , như vậy giá trị này rơi vào khoảng 73 đến 74, với RAWESTIMATEDATA[6][73] = 2003.1804 và RAWESTIMATEDATA[6][74] = 2026.071.

$$2003.1804 \leq \hat{n} \leq 2026.071.$$

Ước lượng chính xác nằm trong khoảng:

$$[2023.1804 - 134.1804, 2026.071 - 131.071] = [1869.0, 1895.0]$$

và để tính toán ước lượng được sửa, chúng ta có thể nội suy giá trị đó, ví dụ, sử dụng tìm kiếm k-nearest neighbor hoặc chỉ là một nội suy tuyến tính

HyperLogLog++ algorithm

Theo các thí nghiệm thực hiện bởi các tác giả của HyperLogLog++, ước lượng n_{lin} được tính theo thuật toán Linear Counting vẫn tốt hơn cho các số lượng phần tử nhỏ so với giá trị được hiệu chỉnh sai số n . Do đó, nếu ít nhất một bộ đếm trống tồn tại, thuật toán sẽ tính toán thêm ước lượng tuyến tính và sử dụng một danh sách các ngưỡng thực nghiệm, có thể tìm thấy trong Bảng 3.4, để chọn xem ước lượng nào nên được ưu tiên. Trong trường hợp như vậy, giá trị được hiệu chỉnh sai số n chỉ được sử dụng khi ước lượng tuyến tính n_{lin} vượt qua ngưỡng κ_m cho m hiện tại. Trong ví dụ (1), khi $m = 2^{10}$, chúng ta tính được giá trị được hiệu chỉnh sai số $n \approx 1886.218$. Để xác định xem chúng ta có nên ưu tiên giá trị này so với ước lượng bằng Linear Counting hay không, chúng ta cần tìm ra số lượng bộ đếm trống Z trong cấu trúc dữ liệu của HyperLogLog++. Vì chúng ta không có giá trị trong ví dụ của chúng ta, hãy giả định rằng $Z = 73$. Do đó, ước lượng tuyến tính là:

$$n_{lin} = 2^{10} \cdot \log\left(\frac{2^{10}}{73}\right) \approx 2704.$$

Tiếp theo, chúng ta so sánh n_{lin} với ngưỡng $\kappa_m = 900$ từ Table 2, mà là

Table 2: Ngưỡng thực nghiệm \varkappa_m cho các giá trị độ chính xác được hỗ trợ

p	m	\varkappa_m
4	2^4	10
5	2^5	20
6	2^6	40
7	2^7	80
8	2^8	220

p	m	\varkappa_m
9	2^9	400
10	2^{10}	900
11	2^{11}	1800
12	2^{12}	3100
13	2^{13}	6500

p	m	\varkappa_m
14	2^{14}	11500
15	2^{15}	20000
16	2^{16}	50000
17	2^{17}	120000
18	2^{18}	350000

HyperLogLog++ algorithm

Algorithm 4: Estimating cardinality with *HyperLogLog++*

Input: Dataset D

Input: Array of m *LogLog* counters with hash function h

Output: Cardinality estimation

$COUNTER[j] \leftarrow 0, j = 0 \dots m - 1$

for $x \in D$ **do**

$i \leftarrow h(x) := (i_0 i_1 \dots i_{63})_2, i_k \in \{0, 1\}$

$j \leftarrow (i_0 i_1 \dots i_{p-1})_2$

$r \leftarrow COUNTER[j] \leftarrow \max(COUNTER[j], r)$

end

$R \leftarrow \sum_{k=0}^{m-1} 2^{-COUNTER[j]}$

$\hat{n} = \alpha_m \cdot m^2 \cdot \frac{1}{R}$

$n \leftarrow \hat{n}$

if $\hat{n} \leq 5m$ **then**

$n \leftarrow \text{CorrectBias}(\hat{n})$

end

HyperLogLog++ algorithm

Độ chính xác của HyperLogLog++ tốt hơn so với HyperLogLog cho một phạm vi lớn của các số lượng phần tử và tương đương tốt cho phần còn lại. Đối với các số lượng phần tử từ 12000 đến 61000, việc hiệu chỉnh sai số cho phép giảm thiểu sai số và tránh một đỉnh sai số khi chuyển đổi giữa các phụ thuộc (sub-algorithms).

Tuy nhiên, vì HyperLogLog++ không cần lưu trữ giá trị băm, chỉ cần một cộng với kích thước tối đa của số lượng số không đầu tiên, yêu cầu bộ nhớ không tăng đáng kể so với HyperLogLog và chỉ yêu cầu $6 \cdot 2^p$ bit. Thuật toán HyperLogLog++ có thể được sử dụng để ước lượng số lượng phần tử

khoảng $7.9 \cdot 10^9$ với một tỷ lệ lỗi điển hình là 1.625%, sử dụng 2.56 KB bộ nhớ.

HyperLogLog++ algorithm

Như đã đề cập trước đó, thuật toán sử dụng phương pháp lấy trung bình ngẫu nhiên và chia tập dữ liệu thành $m = 2^p$ tập con $\{COUNTER[j]\}_{j=0}^{m-1}$, mỗi bộ đếm xử lý thông tin về $\frac{n}{m}$ phần tử. Heule và đồng nghiệp đã nhận thấy rằng đối với $n \ll m$, hầu hết các bộ đếm không bao giờ được sử dụng và không cần phải được lưu trữ, do đó lưu trữ có thể được hưởng lợi từ một biểu diễn thưa thớt. Nếu số lượng phần tử n nhỏ hơn rất nhiều so với m , thì HyperLogLog++ yêu cầu bộ nhớ đáng kể ít hơn so với các phiên bản trước đó.

Thuật toán HyperLogLog++ trong phiên bản thưa thớt chỉ lưu trữ các cặp $(j, COUNTER[j])$, biểu diễn chúng dưới dạng một số nguyên duy nhất bằng cách nối các mẫu bit của chúng. Tất cả các cặp như vậy được lưu trữ trong một danh sách đã sắp xếp duy nhất của các số nguyên. Vì chúng ta luôn tính toán hạng cực đại, nên chúng ta không cần phải lưu trữ các cặp khác nhau có cùng chỉ số, thay vào đó chỉ cần lưu trữ cặp có chỉ số cực đại.

Trong thực tế, để cung cấp trải nghiệm tốt hơn, người ta có thể duy trì một danh sách không được sắp xếp khác để thực hiện các thêm nhanh

Phương pháp thực hiện

Bài toán 1

Phát triển thuật toán để ước lượng số lượng phần tử (cardinality estimation) trong một khoảng thời gian trên một dòng dữ liệu (data stream):

- Bước 1: Xác định khoảng thời gian Đầu tiên, chúng tôi sẽ xác định khoảng thời gian mà chúng tôi muốn đếm số lượng phần tử. Ví dụ, mỗi giờ hoặc mỗi phút.
- Bước 2: Lưu trữ HyperLogLog Tiếp theo, chúng tôi sẽ lưu trữ cấu trúc HyperLogLog cho mỗi khoảng thời gian. Cấu trúc dữ liệu sẽ bao gồm cặp $\langle T_1, HLL_1 \rangle$, trong đó T_1 là thời điểm đại diện cho khung thời gian cụ thể.
- Bước 3: Sử dụng kết quả Cuối cùng, khi cần, chúng tôi có thể truy vấn và sử dụng kết quả từ các cấu trúc HyperLogLog lưu trữ theo khung thời gian để ước lượng số lượng phần tử trong mỗi khoảng thời gian.

Bài toán 2 (1/2)

Mở rộng thuật toán để ước lượng số lượng phần tử trong một khoảng thời gian trên nhiều dòng dữ liệu:

Ví dụ khi chúng ta cần biết có bao nhiêu người dùng đã đăng nhập vào hệ thống vào ngày hôm qua, do dữ liệu người dùng được lưu ở trên nhiều hệ thống như web, application và cũng như trên các bộ phận khác nhau của doanh nghiệp. Khi đó chúng ta sẽ có nhiều nguồn dữ liệu khác nhau và cần một thuật toán để kết hợp các nguồn dữ liệu này để tổng hợp cho ra ước lượng số lượng cuối cùng.

Bài toán 2 (2/2)

- Bước 1: Lưu trữ dữ liệu
 - Lưu trữ dữ liệu theo khung thời gian $\langle T, HLL \rangle$
- Bước 2: Tổng hợp HLL từ các dữ liệu từ nhiều nơi khác nhau:
 $\langle T, HLL_1 \rangle, \langle T, HLL_2 \rangle, \dots, \langle T, HLL_N \rangle$
 - T là khoảng thời gian cần tổng hợp
 - HLL_1 là dữ liệu HyperLogLog trong khoảng thời gian
- Bước 3: Ước lượng số phần tử

$$E = \alpha_m \cdot m^2 \cdot \left(\sum_{j=1}^m 2^{-M[j]} \right)^{-1}$$

- E là ước lượng số lượng phần tử duy nhất
- α_m là hằng số
- m là số lượng register trong cấu trúc HyperLogLog
- $M[j]$ là giá trị của register thứ j

Kế hoạch triển khai

Kế hoạch triển khai

#	Tuần	Nội dung công việc
1	1 - 2	Bài báo liên quan mới nhất và bổ sung cơ sở lý thuyết về các kỹ thuật ước lượng số lượng trên dòng dữ liệu
2	3 - 4	Thu thập dữ liệu, chuẩn hoá và tiền xử lý. Hiện thực bài toán 1 ước lượng số lượng phần tử trên dòng dữ liệu
3	5 - 6	Mở rộng để ước lượng số lượng phần tử trên nhiều dòng dữ liệu. Đánh giá hiệu suất và độ chính xác.
4	7 - 8	Phân tích và so sánh kết quả, đánh giá ưu nhược điểm. Đề xuất phương pháp tối ưu hiệu suất và độ chính xác.
5	9 - 10	Ứng dụng kết quả nghiên cứu. Đề xuất hướng phát triển và nghiên cứu tiếp theo.
6	11 - 12	Đề xuất và đánh giá các giải pháp
7	1 - 14	Tổng hợp kết quả và viết báo cáo

Nội dung dự kiến của luận văn

Chương 1: Giới thiệu. Tầm quan trọng của việc phát triển kỹ thuật đếm số phần tử trên dòng dữ liệu trong ngữ cảnh dữ liệu lớn.

Chương 2: Các công trình nghiên cứu liên quan. Các công trình nghiên cứu liên quan, phương pháp giải quyết vấn đề. Đánh giá tính khả thi của đề tài.

Chương 3: Kiến thức nền tảng. Giới thiệu về tính chất, phương pháp truy vấn và xử lý trên dòng dữ liệu. Giới thiệu về HyperLogLog và nguyên lý hoạt động và đánh giá hiệu suất, độ chính xác trên dòng dữ liệu.

Chương 4: Hiện thực và thử nghiệm. Trong chương này sẽ trình bày chi tiết cách thức hiện thực của từng thuật toán.

Chương 5: Kết quả và đánh giá. Trong chương này sẽ nêu ra các kết quả đạt được của các kỹ thuật, cũng như phương pháp đánh giá dựa trên kết quả thực nghiệm.

Chương 6: Kết luận. Đánh giá ưu điểm và nhược điểm của mô hình và đề xuất hướng nghiên cứu phát triển kỹ thuật đếm số phần tử trong tương lai.

Việc phát triển kỹ thuật đếm số lượng phần tử trên dòng dữ liệu là cần thiết để tối ưu hóa hiệu quả hoạt động, nâng cao trải nghiệm người dùng và hỗ trợ ra quyết định kinh doanh.

Nghiên cứu này góp phần quan trọng vào lĩnh vực khoa học máy tính, đặc biệt trong bối cảnh dữ liệu lớn ngày càng phát triển.



Marianne Durand and Philippe Flajolet.

Loglog counting of large cardinalities.

In *Algorithms-ESA 2003: 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003. Proceedings 11*, pages 605–617. Springer, 2003.



Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier.

Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm.

Discrete mathematics & theoretical computer science, (Proceedings), 2007.



Stefan Heule, Marc Nunkesser, and Alexander Hall.

Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm.

In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 683–692, 2013.



Yousra Chabchoub and Georges Heébrail.

Sliding hyperloglog: Estimating cardinality in a data stream over a sliding window.

In *2010 IEEE International Conference on Data Mining Workshops*, pages 1297–1303. IEEE, 2010.



Otmar Ertl.

Exaloglog: Space-efficient and practical approximate distinct counting up to the exa-scale.

arXiv preprint arXiv:2402.13726, 2024.