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

Outline

Giới thiệu

Giới thiệu

- Ứng dụng và dịch vụ trực tuyến đóng vai trò quan trọng trong cuộc sống hiện đại.
- DAU (**Daily Active Users**) 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.

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 sai số khoảng 2% chỉ dụng 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.

Phát biểu bài toán

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

- 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.
- Thực hiện, phân tích kết quả thí nghiệm, rút ra kết luận và đề xuất hướng nghiên cứu tiếp theo.
- Đóng góp vào sự phát triển của công nghệ dữ liệu lớn, mở ra tiềm năng ứng dụng rộng lớn trong nhiều lĩnh vực.

Giới hạn, đối tượng nghiên cứu

Đối tượng nghiên cứu:

- Dòng dữ liệu dạng **văn bản** có chứa nhiều phần tử cần đếm.
  - userID, IP address, words, etc
- Các kỹ thuật đếm số lượng phần tử.

Giới hạn:

- 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.
  - Hình ảnh, âm thanh ...

Cơ sở lý thuyết

LogLog Algorithm

Algorithm LogLog($\mathfrak{M}$: Multiset of hashed values; $m \equiv 2^k$)
Initialize $M^{(1)}, \ldots, M^{(m)}$ to 0;
let $\rho(y)$ be the rank of first 1-bit from the left in y;
for $x = b_1 b_2 \cdots \in \mathfrak{M}$ do
set $j := \langle b_1 \cdots b_k \rangle_2$ (value of first k bits in base 2)
set $M^{(j)} := \max(M^{(j)}, \rho(b_{k+1} b_{k+2} \cdots)$;
return $E := \alpha_m m 2^{\frac{1}{m} \sum_j M^{(j)}}$ as cardinality estimate.

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%.

HyperLogLog algorithm

- Đã được đề xuất bởi Philippe Flajolet, Eric Fusy, Olivier Gandouet và Frederic Meunier vào năm 2007 [2].
- Sử dụng hàm băm 32-bit và hàm đánh giá có các sửa lỗi **bias** khác nhau.
- 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 $p \in 4...16$.
- Sử dụng trung bình điều hoà (**hamonic** mean) thay vì sử dụng trung bình hình học (geometric mean) như phiên bản gốc LogLog

HyperLogLog algorithm

Let $h : \mathcal{D} \to [0,1] \equiv \{0,1\}^\infty$ *hash data from domain* $\mathcal{D}$ *to the binary domain.*
Let $\rho(s)$*, for* $s \in \{0,1\}^\infty$*, be the position of the leftmost 1-bit* ($\rho(0001\cdots) = 4$).

Algorithm HYPERLOGLOG (**input** $\mathcal{M}$: multiset of items from domain $\mathcal{D}$).
assume $m = 2^b$ with $b \in \mathbb{Z}_{>0}$;
initialize a collection of m registers, $M[1], \ldots, M[m]$, to $-\infty$;

for $v \in \mathcal{M}$ **do**
    set $x := h(v)$;
    set $j = 1 + \langle x_1 x_2 \cdots x_b \rangle_2$; {*the binary address determined by the first b bits of*
    set $w := x_{b+1} x_{b+2} \cdots$; **set** $M[j] := \max(M[j], \rho(w))$;

compute $Z := \left(\sum_{j=1}^{m} 2^{-M[j]} \right)^{-1}$; {*the "indicator" function*}

return $E := \alpha_m m^2 Z$ with α_m as given by Equation (3).

Với α_m

$$\alpha_m = \left(m \int_0^\infty \left(\log_2 \left(\frac{2+x}{1+x} \right) \right)^m dx \right)^{-1} \qquad (3)$$

HyperLogLog algorithm

- Tương tự LogLog, sai số tiêu chuẩn δ:

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

- Yêu cầu bộ nhớ không tăng tuyến tính theo số lượng phần tử, phân bổ $(M = p)$ bit cho các giá trị băm và có tổng cộng $m = 2^p$ bộ đếm, bộ nhớ cần thiết là:

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

- Sử dụng hàm băm 32-bit và độ chính xác $p \in 4...16$, yêu cầu bộ nhớ 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, **Redis** duy trì cấu trúc dữ liệu *HyperLogLog* của **12 KB** để xấp xỉ các định lượng $\delta \approx 0.81.\%$.

HyperLogLog: các thư viện và công cụ nổi bật

- **Apache DataSketches:** một thư viện Java cung cấp các thuật toán xác suất và thống kê. Được sử dụng rộng rãi trong các hệ thống Big Data như Apache Druid, Apache Kafka và Apache Hive.
- **Redis:** hệ thống lưu trữ dữ liệu dạng key-value phổ biến, cung cấp cấu trúc dữ liệu HyperLogLog tích hợp sẵn. Sử dụng các lệnh như PFADD, PFCOUNT và PFMERGE để làm việc với HLL.
- **Google BigQuery:** sử dụng HyperLogLog++ cho các chức năng thống kê và phân tích dữ liệu lớn.
- **PostgreSQL:** Extension postgreSQL-hll giúp thực hiện các truy vấn với số lượng phần tử duy nhất một cách hiệu quả.
- **Amazon Redshift:** hỗ trợ HyperLogLog để tối ưu hóa các truy vấn thống kê và giảm thiểu dung lượng bộ nhớ cần thiết.
- **Apache Flink:** một nền tảng xử lý luồng dữ liệu phân tán, có tích hợp HyperLogLog để xử lý các phép tính phức tạp trên dữ liệu luồng.

Ví dụ: Redis (1/3)

Redis là một hệ thống lưu trữ dữ liệu dạng key-value phổ biến và hỗ trợ nhiều cấu trúc dữ liệu mạnh mẽ, trong đó có HyperLogLog. Redis cung cấp các lệnh chuyên biệt để làm việc với HyperLogLog, bao gồm PFADD, PFCOUNT và PFMERGE.

Các lệnh cơ bản của HyperLogLog trong Redis:

- **PFADD:** Thêm các phần tử vào HyperLogLog.
- **PFCOUNT:** Ước lượng số phần tử duy nhất trong HyperLogLog.
- **PFMERGE:** Hợp nhất nhiều HyperLogLog thành một.

Ví dụ: Redis (2/3)

Giả sử chúng ta có một ứng dụng web và muốn theo dõi số lượng người dùng duy nhất truy cập vào website mỗi ngày.
Các lệnh cơ bản của HyperLogLog trong Redis:

- **Cài đặt Redis:**
  - sudo apt-get update
  - sudo apt-get install redis-server
- **Khởi động Redis server:**
  - redis-server
- **Kết nối tới Redis:**
  - redis-cli

Ví dụ: Redis (3/3)

- **Thêm người dùng**:

```
PFADD unique_visitors:2024-05-25T08 user1 user2
    user3
PFADD unique_visitors:2024-05-25T08 user2 user4
PFADD unique_visitors:2024-05-25T09 user5 user6
PFADD unique_visitors:2024-05-25T10 user1 user7
```

- **Ước lượng số người dùng trong giờ 08:00 ngày 2024-05-25**:

```
PFCOUNT unique_visitors:2024-05-25T08
```

- **Giả sử chúng ta muốn hợp nhất dữ liệu người dùng duy nhất từ ba giờ khác nhau (08:00, 09:00, và 10:00).**:

```
PFMERGE unique_visitors:2024-05-25:morning
    unique_visitors:2024-05-25T08
    unique_visitors:2024-05-25T09
    unique_visitors:2024-05-25T10
PFCOUNT unique_visitors:2024-05-25:morning
```

Cách HyperLogLog lưu trữ dữ liệu trong Redis

Redis sử dụng "sparse representation" cho các bộ đếm nhỏ và "dense representation" cho các bộ đếm lớn hơn.
Dung lượng bộ nhớ phụ thuộc vào số lượng thanh ghi (registers), và mỗi thanh ghi lưu trữ thông tin về vị trí của bit đầu tiên là 1 trong chuỗi băm.

Số lượng thanh ghi (m):

- $m = 2^p$, trong đó p là số bit để xác định số thanh ghi.
- Giá trị mặc định p trong Redis là **14** => có $2^{14} = 16384$ thanh ghi.

Dung lượng bộ nhớ cần thiết

- Mỗi thanh ghi cần **6 bit** để lưu trữ vị trí của bit đầu tiên là 1.
- Tổng dung lượng bộ nhớ cần thiết cho HyperLogLog trong Redis có thể tính theo công thức (với m = 16384 thanh ghi):
  Memory = 16384 x 6 bits = 98304 bits = 12288 bytes = **12 KB**

Tính toán dung lượng cần thiết cho nhiều tập hợp HLL

Nếu bạn muốn lưu trữ nhiều tập hợp HyperLogLog trong Redis, ví dụ như theo dõi số người dùng duy nhất theo giờ, cần nhân dung lượng bộ nhớ của một tập hợp HLL với số lượng tập hợp bạn có.
Giả sử bạn muốn lưu trữ dữ liệu người dùng duy nhất cho mỗi giờ trong một ngày (24 giờ):

$$\text{Total Memory} = 12 \text{ KB} \times 24 = \mathbf{288 \text{ KB}}$$

Nếu bạn muốn lưu trữ dữ liệu theo từng giờ cho nhiều ngày, bạn chỉ cần nhân thêm số lượng ngày:

$$\text{Total Memory for 30 days} = 288 \text{ KB/day} \times 30 = 8640 \text{ KB} = \mathbf{8.64 \text{ MB}}$$

Tối ưu hóa dung lượng bộ nhớ

- Redis sử dụng "sparse representation" cho các bộ đếm nhỏ hơn, giúp tiết kiệm bộ nhớ khi số lượng phần tử trong HyperLogLog còn ít.
- Khi số lượng phần tử tăng, Redis chuyển sang "dense representation" để đảm bảo độ chính xác và hiệu suất.
- Redis tự động chuyển đổi giữa hai biểu diễn này dựa trên số lượng phần tử và mức độ lấp đầy của các thanh ghi, giúp tối ưu hóa việc sử dụng bộ nhớ và đảm bảo độ chính xác cao trong ước lượng số lượng phần tử duy nhất.

Sparse Representation

Đặc điểm:

- **Tiết kiệm bộ nhớ**: Sparse Representation được thiết kế để tiết kiệm bộ nhớ khi số lượng phần tử trong tập hợp còn ít.
- **Cấu trúc nén**: Lưu trữ các cặp (index, value) để chỉ lưu trữ thông tin cần thiết về các thanh ghi được cập nhật.
- **Hiệu quả cho các tập hợp nhỏ**: Rất hiệu quả khi số lượng phần tử còn ít, vì không cần lưu trữ toàn bộ 16384 thanh ghi.

Ước lượng số phần tử:

- **Cơ chế hoạt động**: Các giá trị băm của phần tử được ánh xạ tới một trong 16384 thanh ghi, nhưng chỉ các thanh ghi có giá trị khác 0 mới được lưu trữ.
- **Độ chính xác**: Độ chính xác của ước lượng trong sparse representation tương tự như trong dense representation khi số lượng phần tử còn nhỏ, vì các thanh ghi được quản lý chặt chẽ và thông tin được lưu trữ một cách nén.

Dense Representation

Đặc điểm:

- **Sử dụng bộ nhớ cố định**: Khi số lượng phần tử lớn, HyperLogLog chuyển sang Dense Representation, lưu trữ toàn bộ mảng 16384 thanh ghi với mỗi thanh ghi sử dụng 6 bit.
- **Hiệu quả cho các tập hợp lớn**: Dense Representation trở nên hiệu quả hơn khi số lượng phần tử tăng, vì việc nén không còn mang lại lợi ích về bộ nhớ so với việc lưu trữ toàn bộ thanh ghi.

Ước lượng số phần tử:

- **Cơ chế hoạt động**: Tương tự như sparse representation, nhưng toàn bộ mảng thanh ghi được lưu trữ và sử dụng để tính toán ước lượng.
- **Độ chính xác**: Độ chính xác của ước lượng trong dense representation cao hơn khi số lượng phần tử lớn, vì nó có thể quản lý thông tin của tất cả các thanh ghi mà không cần nén.

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 $\langle T, HLL_1 \rangle$, trong đó T 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: $\langle T, HLL_1 \rangle, \langle T, HLL_2 \rangle, ..., \langle T, HLL_N \rangle$
  - 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 suất phương pháp tối ưu hiệu suất và độ chính xác.
5	9 - 10	Ứng dụng kết quả nghiên cứu. Đề suất 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.

Tài liệu tham khảo i

- M. Durand and P. Flajolet, "Loglog counting of large cardinalities," in *Algorithms-ESA 2003: 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003. Proceedings 11*, pp. 605–617, Springer, 2003.
- P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," *Discrete mathematics & theoretical computer science*, no. Proceedings, 2007.
- S. Heule, M. Nunkesser, and A. 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*, pp. 683–692, 2013.

Questions?