**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**
**TRƯỜNG ĐẠI HỌC BÁCH KHOA**
**KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**

# NGHIÊN CỨU PHÁT TRIỂN KỸ THUẬT ĐẾM SỐ PHẦN TỬ TRÊN DÒNG DỮ LIỆU

## ĐỀ CƯƠNG LUẬN VĂN THẠC SĨ

*Học viên:* **LÊ ANH QUỐC**
*ID:* **2070428**

HỒ CHÍ MINH CITY

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**
**TRƯỜNG ĐẠI HỌC BÁCH KHOA**
**KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**

---

# NGHIÊN CỨU PHÁT TRIỂN KỸ THUẬT ĐẾM SỐ PHẦN TỬ TRÊN DÒNG DỮ LIỆU

## ĐỀ CƯƠNG LUẬN VĂN THẠC SĨ

NGÀNH: KHOA HỌC MÁY TÍNH

MÃ NGÀNH: **8480101**

**NGƯỜI HƯỚNG DẪN KHOA HỌC**
**PGS. TS. THOẠI NAM**

*Học viên:* **LÊ ANH QUỐC**
*ID:* **2070428**

HỒ CHÍ MINH CITY

# Content

# 1 GIỚI THIỆU ĐỀ TÀI, MỤC TIÊU VÀ ĐỐI TƯỢNG NGHIÊN CỨU

## 1.1 Tính cấp thiết và lý do chọn đề tài

Ngày nay, các ứng dụng và dịch vụ trực tuyến đóng vai trò ngày càng quan trọng trong cuộc sống của con người. Chúng ta sử dụng mạng xã hội để kết nối với bạn bè và chia sẻ thông tin, mua sắm trực tuyến để tiết kiệm thời gian và tiền bạc, hay xem phim và chơi game trực tuyến để giải trí. Để đánh giá hiệu quả hoạt động của các ứng dụng và dịch vụ này, một trọng những chỉ số quan trọng nhất là số lượng người dùng hoạt động.

Việc theo dõi số lượng người dùng hoạt động trong một khoảng thời gian nhất định trên một dòng dữ liệu (data stream) là một yêu cầu quan trọng đối với nhiều ứng dụng và dịch vụ trực tuyến, hiệu quả của các chiến dịch marketing, và hỗ trợ ra quyết định kinh doanh.
Ví dụ, trong các ứng dụng mạng xã hội, số lượng người dùng hoạt động cho thấy mức độ tương tác và sự quan tâm của người dùng đối với nền tảng. Trong các dịch vụ thương mại điện tử, số lượng người dùng hoạt động cho thấy hiệu quả và các chiến dịch quảng cáo và khuyến mãi.
Tuy nhiên, việc đếm số lượng người dùng không phải là một nhiệm vụ đơn giản, đặc biệt là khi dữ liệu lớn và tốc độ truy cập cao. Các phương pháp truyền thống như lưu trữ và truy vấn trực tiếp vào cơ sở dữ liệu có thể gặp nhiều hạn chế về hiệu suất và khả năng mở rộng.

Trong nhiều trường hợp, cần phải tổng hợp số lượng người dùng trên nhiều dòng dữ liệu khác nhau. Việc này giúp có được bức tranh toàn cảnh về hoạt động của người dùng trên toàn hệ thống, từ đó đưa ra các phân tích và đánh giá chính xác hơn.
Ví dụ, trong hệ thống thương mại điện tử, cần tổng hợp số lượng người dùng từ các trang web, ứng dụng di động và API khác nhau để có được số lượng người dùng hoạt động thực tế trên toàn hệ thống. Tuy nhiên, việc tổng hợp dữ liệu từ nhiều nguồn khác nhau có thể gặp thách thức về đồng bộ hóa dữ liệu, xử lý dữ liệu bị thiếu hoặc lỗi, và đảm bảo tính nhất quán của kết quả.

Ngoài ra, có thể cần phải đếm số lượng người dùng trên nhiều khoảng thời gian khác nhau trên một hoặc nhiều dòng dữ liệu khác nhau. Việc này giúp phân tích chi tiết hơn hoạt động của người dùng theo thời gian, theo khu vực hoặc theo tiêu chí khác.
Ví dụ, trong một ứng dụng phát trực tiếp, cần đếm số lượng người dùng hoạt động theo giờ hoặc từng phân đoạn chương trình để đánh giá mức độ quan tâm của người xem. Tuy nhiên, việc phân chia và xử lý dữ liệu theo nhiều đoạn có thể làm tăng độ phức tạp của thuật toán và ảnh hưởng đến hiệu suất của hệ thống. Do đó, cần phải có một giải pháp đếm số lượng phần tử trên dòng dữ liệu đạt hiệu suất cao và tin cậy, từ đó có thể ứng dụng rộng rãi trong các hệ thống khác nhau như mạng xã hội, thương mại điện tử, chương trình phát trực tiếp, hệ thống giám sát và hệ thống giao thông thông minh.

# 2 CÁC CÔNG TRÌNH NGHIÊN CỨU LIÊN QUAN

- **LogLog** - [2] Thuật toán LogLog cho phép ước lượng số lượng từ vựng khác nhau trong toàn bộ tác phẩm của Shakespeare chỉ trong một lần quét và với độ chính xác cỡ vài phần trăm, sử dụng một lượng bộ nhớ phụ nhỏ. Phiên bản cơ bản đã được xác minh qua phân tích toàn diện và có phiên bản tối ưu hóa có khả năng song song.
- **HyperLogLog** - [4] Thuật toán HYPERLOGLOG là một thuật toán xác suất gần tối ưu, được thiết kế để ước lượng số lượng các phần tử khác nhau trong các tập dữ liệu rất lớn. Sử dụng bộ nhớ phụ có kích thước m đơn vị, HYPERLOGLOG thực hiện một lần quét qua dữ liệu và tạo ra một ước lượng về số lượng phần tử khác nhau với độ chính xác tương đối là khoảng $\frac{1.04}{\sqrt{m}}$. Thuật toán này có khả năng ước lượng số lượng phần tử lớn hơn $10^9$ với độ chính xác khoảng 2% chỉ sử 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 windown).
- **HyperLogLog++** - [5] Bài báo giới thiệu các cải tiến cho thuật toán HyperLogLog nhằm giảm yêu cầu bộ nhớ và tăng độ chính xác đối với một phạm vi quan trọng của số phân biệt. Cải tiến này đã được triển khai và đánh giá trên một hệ thống tại Google, so sánh với phiên bản gốc của HyperLogLog. Như HyperLogLog, thuật toán cải tiến này có khả năng song song hoàn hảo và tính toán ước lượng độ phân biệt trong một lượt đi duy nhất.
- **Sliding HyperLogLog** - [1] Bài báo giới thiệu một thuật toán mới ước lượng số lượng luồng hoạt động trong dòng dữ liệu, sử dụng cơ chế cửa sổ trượt kết hợp với thuật toán HyperLogLog. Thuật toán này có độ chính xác cao, lỗi tiêu chuẩn khoảng $\frac{1.04}{\sqrt{m}}$, với m là số lượng thanh ghi trong bộ nhớ. Dù cần bộ nhớ bổ sung so với HyperLogLog, tổng bộ nhớ cần thiết không vượt quá $5m\ln(\frac{n}{m})$ byte, với n là số luồng thực sự trong cửa sổ trượt. Kết quả lý thuyết được xác minh trên cả dữ liệu thực và tổng hợp.
- **ExaLogLog** - [3] ExaLogLog 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.

# 3 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:

# 4 Mục tiêu, Giới hạn và đối tượng nghiên cứu

## 4.1 Mục tiêu

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.

## 4.2 Giới hạn

- 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.

## 4.3 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.

# 5 Cơ sở lý thuyết

Các thuật toán xác suất phổ biến nhất để ước lượng số lượng được sử dụng trong thực tế là họ các thuật toán LogLog bao gồm thuật toán *LogLog*, được đề xuất bởi Marianne Durand và Philippe Flajolet vào năm 2003 [2], và các kế thừa của nó *HyperLogLog* và *HyperLogLog++*.
Các thuật toán này sử dụng một phương pháp tương tự như thuật toán Đếm Xác Suất trong việc ước lượng số lượng $n$ bằng cách quan sát số lượng lớn nhất của các số không dấu hàng đầu trong biểu diễn nhị phân của các giá trị. Tất cả chúng đều yêu cầu một bộ nhớ phụ trợ và thực hiện một lần duyệt qua dữ liệu để tạo ra một ước lượng về số lượng.

Mỗi phần tử trong tập dữ liệu được tiền xử lý bằng cách áp dụng một hàm băm $h$ chuyển đổi các phần tử thành số nguyên phân bố đều đặn đủ trên một phạm vi scala $\{0, 1, ..., 2^M - 1\}$ hoặc, tương đương, trên tập hợp các chuỗi nhị phân có độ dài M:

$$h(x) = j = \sum_{k=0}^{M-1} j_k \cdot 2^k := (i_0 i_1 ... i_{M-1})_2 \, , i_k \in \{0, 1\}. \qquad (5.1)$$

Đầu tiên, chúng ta chia tập dữ liệu ban đầu hoặc dòng dữ liệu đầu vào thành một số tập con, mỗi tập con này được lập chỉ mục bởi một trong $m$ bộ đếm đơn giản. Sau đó, theo phương pháp trung bình ngẫu nhiên, vì có một hàm băm đơn giản, chúng ta chọn bộ đếm cho phần tử cụ thể $x$ bằng cách sử dụng một phần của giá trị băm của nó $h(x)$, trong khi phần còn lại được sử dụng để cập nhật bộ đếm tương ứng.

Tất cả các thuật toán được thảo luận ở đây dựa trên việc quan sát các mẫu $0^k1$ xuất hiện ở đầu của các giá trị cho bộ đếm cụ thể, và gán mỗi mẫu với chỉ số của nó, gọi là hạng (rank). Hạng tương đương với vị trí bit 1 đầu tiên từ trái qua phải trong biểu diễn nhị phân của giá trị băm của phần tử được lập chỉ mục và có thể được tính bằng công thức:

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

Mỗi bộ đếm đơn giản xây dựng quan sát về số lượng của riêng mình dựa trên các hạng đã nhìn thấy, ước lượng cuối cùng của số lượng được tạo ra từ quan sát bằng một hàm đánh giá.

Liên quan đến lưu trữ, thuật toán *LogLog* đề xuất một giải pháp tiết kiệm lưu trữ cùng với một hàm đánh giá và phương pháp hiệu chỉnh sai lệch tốt hơn.

## 5.1 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 $\pm 1.87$ 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 $S_0, S_1, ..., S_{m-1}$, 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.

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

và phần còn lại (M-p) bit được lập chỉ mục vào bộ đếm tương ứng COUNTER[j] để tính hạng và quan sát $R_j$ theo công thức (3.9).

Dưới sự phân phối công bằng, mỗi tập con nhận $\frac{n}{m}$ phần tử, do đó quan sát $R_j$ từ các bộ đếm $\{COUNTER[j]\}_{j=0}^{m-1}$ có thể cung cấp một dấu hiệu về giá trị của $\log_2 \frac{n}{m}$, và bằng cách sử dụng trung bình số học của chúng với một số sự hiệu chỉnh, chúng ta có thể giảm thiểu phương sai của một quan sát duy nhất:

$$n = \alpha_m \cdot m \cdot 2^{\frac{1}{m}\sum\limits_{j=0}^{m-1} R_j},$$

với $\alpha_m = \left(\Gamma\left(-\frac{1}{m}\right) \cdot \frac{1-2^{\frac{1}{m}}}{\log_2}\right)^m$, $\Gamma(\cdot)$ là gamma function.

Tuy nhiên, đối với hầu hết các trường hợp thực tế, $m \geq 64$ là đủ để chỉ sử dụng $\alpha_m \approx 0.39701$.

**Algorithm 1:** Estimatin cardinality with *LogLog*

**Input:** Dataset D
**Input:** Array of $m$ *LogLog* counters with hash function $h$
**Output:** Cardinality estimation
$COUNTER[j] \leftarrow 0$, $j = 0...m-1$
**for** $x \in D$ **do**
  $i \leftarrow$ h(x) := $(i_0 i_1 ... i_{M-1})_2$, $i_k \in \{0,1\}$
  $j \leftarrow (i_0 i_1 ... i_{M-1})_2$
  $r \leftarrow$ rank($(i_p i_{p+1} ... i_{M-1})_2$)
  $COUNTER[j] \leftarrow$ max(COUNTER[j],r)
**end**
$R \leftarrow \frac{1}{m} \sum\limits_{k=0}^{m-1}$ COUNTER[j]
**return** $\alpha_m \cdot m \cdot 2^R$

Sai số tiêu chuẩn $\delta$ của thuật toán *LogLog* có mối quan hệ nghịch với số lượng bộ đếm sử dụng $m$ và có thể được xấp xỉ gần như là:

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

Do đó, với $m = 256$, sai số tiêu chuẩn là khoảng 8% và với $m = 1024$, 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 đến 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.
$log_2\left(log_2 2^{20}\right) \approx 4.32$, do đó, chỉ cần phân bổ khoảng 5 bit cho mỗi ngăn (tức là một giá trị nhỏ hơn 32). Do đó, để ước lượng định lượng lên đến khoảng $10^9$ với sai số tiêu chuẩn là 4%, thuật toán yêu cầu 1024 ngăn với 5 bit mỗi ngăn, tức là tổng cộng 640 byte.

## 5.2 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 [4]. 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 $p \in 4...16$.

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):

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

với

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

Các giá trị xấp xỉ của $\alpha_m$ có thể được tìm thấy trong Bảng 1.

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:** $\alpha_m$ for most used values of $m$

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

Tuy nhiên, ước lượng **n̂** 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:

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

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, đụng độ 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.

Dưới đây là toàn bộ thuật toán *HyperLogLog*.

**Algorithm 2:** Estimatin cardinality with *HyperLogLog*

**Input:** Dataset D
**Input:** Array of $m$ *LogLog* counters with hash function $h$
**Output:** Cardinality estimation
$COUNTER[j] \leftarrow 0,\ j = 0...m-1$
**for** $x \in D$ **do**
  $i \leftarrow$ h(x) $:= (i_0 i_1 ... i_{M-1})_2,\ i_k \in \{0,1\}$
  $j \leftarrow (i_0 i_1 ... i_{M-1})_2$
  $r \leftarrow$ rank$((i_p i_{p+1} ... i_{M-1})_2)$
  $COUNTER[j] \leftarrow$ max(COUNTER[j],r)
**end**
$R \leftarrow \frac{1}{m} \sum\limits_{k=0}^{m-1}$ COUNTER[j]
$\hat{n} = \alpha_m \cdot m^2 \cdot \frac{1}{R}$
$n \leftarrow \hat{n}$
**if** $\hat{n} \le \frac{5}{2} m$ **then**
  $Z \leftarrow \underset{j=0...m-1}{count} (COUNTER[j] == 0)$
  **if** $Z \neq 0$ **then**
    $n \leftarrow m \cdot \log\left(\frac{m}{Z}\right)$
  **end**
**end**
**else if** $\hat{n} > \frac{1}{30} 2^{32}$ **then**
  $n \leftarrow -2^{32} \cdot \log\left(1 - \frac{n}{2^{32}}\right)$
**end**
**return** $n$

Tương tự như thuật toán *LogLog*, có một sự đánh đổi rõ ràng giữa sai số tiêu chuẩn $\delta$ 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 $p \in 4...16$, 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 định lượng cho các bộ dữ liệu nhỏ, nó vẫn đánh giá quá cao các định lượng thực tế trong các trường hợp như vậy.

Các biến thể của thuật toán $HyperLogLog$ được triển khai trong các cơ sở dữ liệu nổi tiếng như Amazon Redshift, Redis, Apache CouchDB, Riak và các hệ thống khác.

### 5.3 HyperLogLog++ algorithm

Sau một thời gian, vào năm 2013 [1], 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 $2^64 \approx 1.8 \cdot 10^{19}$, đụng độ 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 định dựa trên kinh nghiệm để cải thiện việc sửa lỗi sai trong thuật toán - các mảng ước lượng định lượng thô $RAWESTIMATEDATA$ và các sai lệch liên quan $BIASDATA$. cung cấp một mảng gồm 200 điểm nội suy, lưu trữ ước lượng thô trung bình được đo tại điểm này qua 5000 tập dữ liệu khác nhau. $BIASDATA$ chứa khoảng 200 sai lệch đã được đo lường tương ứng với $RAWESTIMATEDATA$. Cả hai mảng đều có chỉ số bắt đầu từ 0 và chứa các giá trị được tính toán trước cho tất cả các độ chính xác được hỗ trợ $p \in 4...18$, trong đó chỉ số 0 trong các mảng tương ứng với các giá trị độ chính xác 4. Như một ví dụ, đối với $m = 2^{10} and p = 10$ dữ liệu cần thiết có thể được tìm thấy trong $RAWESTIMATEDATA[6]$ và $BIASDATA[6]$.

Quy trình sửa lỗi sai trong thuật toán $HyperLogLog++$ có thể được hình thức như sau.

**Algorithm 3:** Correcting bias in *HyperLogLog++*

**Input:** Estimate $\hat{n}$ with precicion $p$
**Output:** Bias-corrected cardinality estimate
$n_{low} \leftarrow 0, n_{up} \leftarrow 0, j_{low} \leftarrow 0, j_{up} \leftarrow 0$
**for** $j \leftarrow 0 to length(RAWESTIMATEDATA[p-4])$ **do**
  **if** $RAWESTIMATEDATA[p-4][j] \geq \hat{n}$ **then**
    $j_{low} \leftarrow j-1, j_{up} \leftarrow j$
    $n_{low} \leftarrow RAWESTIMATEDATA[p-4][j_{low}]$
    $n_{up} \leftarrow RAWESTIMATEDATA[p-4][j_{up}]$
    break
  **end**
**end**
$b_{low} \leftarrow BIASDATA[p-4][j_{low}]$
$b_{up} \leftarrow BIASDATA[p-4][j_{up}]$
$y = interpolate\left((n_{low}, n_{low} - b_{low}), (n_{up}, n_{up} - b_{up})\right)$
**return** $y(\hat{n})$

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

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

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

$y(x) = a \cdot x + b, where y(2003.1804) = 1869.0 and y(2026.071) = 1895.0.$
Do đó, bằng cách tính toán đơn giản, đường nội suy là

$$y = 1.135837 \cdot x - 406.28725,$$

và giá trị được nội suy cho ước lượng định lượng của chúng ta là

$$n = y(\hat{n}) = y(2018.34) \approx 1886.218.$$

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 $\varkappa_m$ 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 $\varkappa_m = 900$ từ Table 2, mà là rất thấp so với giá trị tính toán, do đó, chúng ta ưu tiên ước lượng được hiệu chỉnh sai số $n$ so với ước lượng của Linear Counting $n_{lin}$.

**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 |

Thuật toán hoàn chỉnh HyperLogLog++ được thể hiện dưới đây.

**Algorithm 4:** Estimating cardinality with *HyperLogLog++*

**Input:** Dataset D
**Input:** Array of $m$ *LogLog* counters with hash function $h$
**Output:** Cardinality estimation
$COUNTER[j] \leftarrow 0, j = 0...m-1$
**for** $x \in D$ **do**
  $i \leftarrow h(x) := (i_0 i_1 ... i_{63})_2, i_k \in \{0, 1\}$
  $j \leftarrow (i_0 i_1 ... i_{p-1})_2$
  $r \leftarrow COUNTER[j] \leftarrow \max(COUNTER[j], r)$
**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**
  $n \leftarrow$ **CorrectBias**$(\hat{n})$
**end**
$Z \leftarrow \underset{j=0...m-1}{\textbf{count}}(COUNTER[j] = 0)$
**if** $Z \neq 0$ **then**
  $n_{lin} \leftarrow m \cdot \log \frac{m}{Z}$
  **if** $n_{lin} \leq \varkappa_m then$ **then**
    $n \leftarrow n_{lin}$
  **end**
**end**
**return** $n$

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-algorithns).

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.

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 cần phải được sắp xếp và hợp nhất định kỳ vào danh sách chính. Nếu một danh sách như vậy yêu cầu nhiều bộ nhớ hơn so với biểu diễn dày đặc của các bộ đếm, nó có thể dễ dàng chuyển đổi thành dạng dày đặc. Ngoài ra, để làm cho biểu diễn thưa thớt thậm chí còn thân thiện với không gian hơn, thuật toán HyperLogLog++ đề xuất các kỹ thuật nén khác nhau bằng cách sử dụng mã hóa độ dài biến và mã hóa sự khác biệt cho các số nguyên, do đó chỉ lưu trữ cặp đầu tiên và sự khác biệt từ giá trị đó.

Hiện nay, thuật toán HyperLogLog++ được rộng rãi sử dụng trong nhiều ứng dụng phổ biến, bao gồm Google BigQuery và Elasticsearch.

## Kết luận

Trong chương này, chúng ta đã đi qua các phương pháp xác suất khác nhau để tính toán các phần tử duy nhất trong các tập dữ liệu lớn. Chúng ta đã thảo luận về những khó khăn xuất hiện trong các nhiệm vụ ước lượng số lượng phần tử và tìm hiểu về một giải pháp đơn giản có thể ước lượng các tập hợp có số lượng phần tử nhỏ khá tốt. Hơn nữa, chúng ta đã nghiên cứu về họ các thuật toán dựa trên việc quan sát các mẫu nhất định trong các biểu diễn băm của các phần tử từ tập dữ liệu, theo sau bởi nhiều cải tiến và sửa đổi đã trở thành tiêu chuẩn ngành công nghiệp ngày nay để ước lượng số lượng phần tử của hầu hết các loại.

# 6 PHƯƠNG PHÁP THỰC HIỆN

## 6.1 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):

Trong phương pháp này, chúng tôi sẽ trình bày quá trình phát triển thuật toán sử dụng HyperLogLog để ước lượng số lượng phần tử trên một dòng dữ liệu. Bằng cách sử dụng cấu trúc dữ liệu HyperLogLog thep khung thời gian, ví dụ như mỗi giờ hoặc mỗi phút và các kỹ thuật tối ưu, chúng tôi xây dựng một thuật toán hiệu quả và chính xác để đếm số lượng phần tử duy nhất trong dữ liệu dòng.

- 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.

## 6.2 Bài toán 1: Mở rộng thuật toán để ước lượng số lượng phần tử trong một khoảng thời gian trên nhiều dòng dữ liệu:

Trong phương pháp này, chúng tôi sẽ sẽ mở rộng thuật toán 1 để ước lượng 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.

- Bước 1: Tổng hợp dữ liệu Đầu tiên, chúng ta đã lưu trữ dữ liệu trên một dòng dữ liệu như thuật toán ở trên.
- Bước 2: Tổng hợp HyperLogLog Tiếp theo, chúng tôi sẽ sẽ tiến hành tổng hợp các dữ liệu từ nhiều nơi khác nhau $\langle T_1, HLL_1\rangle, \langle T_2, HLL_2\rangle, ..., \langle T_N, HLL_N\rangle$, trong đó $T_1, T_2, ..., T_N$ là các khoảng thời gian giống nhau nên $T_1 = T_2 = T_N$ và đặt chung là $T$, và $HLL_1$ là dữ liệu HyperLogLog trong khoảng thời gian, ví dụ từ 12:00 ngày hôm qua cho đến 12:00 ngày hôm nay.
- Bước 3: Ước lượng số phần tử Cuối cùng, sau khi đã tổng hợp dữ liệu thành một cấu trúc HyperLogLog mới ta tiến hành ước lượng số phần tử theo công thức:

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

  Trong đó: $E$ là ước lượng số lượng phần tử duy nhất, $\alpha_m$ là hằng số, được chọn tùy thuộc vào số lượng bộ nhớ sử dụng. $m$ là số lượng register trong cấu trúc HyperLogLog, $M[j]$ là giá trị của register thứ j.

# 7 KẾ HOẠCH TRIỂN KHAI

Trong thời gian hiện thực luận văn, tôi dự định triển khai công việc với nội dung như sau:

Thời gian dự kiến làm luận văn là từ ngày 05/09/2024 đến 15/12/2024

| # | Tuần | Nội dung công việc |
|---|---|---|
| 1 | 1 - 2 | Tìm hiểu thêm các bài báo liên quan mới nhất và bổ sung cơ sở lý thuyết<br>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<br>ước lượng số lượng phần tử trên dòng dữ liệu |
| 3 | 5 - 6 | Hiện thực bài toán mở rộng để ước lượng số lượng phần tử trong một khoảng thời gian<br>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 của từng kỹ thuật ước lượng.<br>Đề suất phương pháp hay tối ưu hiệu suất và độ chính xác. |
| 5 | 9 - 10 | Ứng dụng kết quả nghiên cứu.<br>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 |

# 8 NỘI DUNG DỰ KIẾN CỦA LUẬN VĂN

Nội dung báo cáo của luận văn dự kiến sẽ bao gồm các phần như sau:

**Chương 1: Giới thiệu**. Chương này nhấn mạnh về tầm quan trọng của việc phát triển kỹ thuật đếm số phần tử trên dòng dữ liệu trong ngữ cảnh dữ liệu lớn. Qua đó, ta sẽ hiểu rõ hơn về phạm vi và mục tiêu của nghiên cứu, cũng như cách tiếp cận vấn đề.

**Chương 2: Các công trình nghiên cứu liên quan.** Trong chương này, sẽ được trình bày các công trình nghiên cứu liên quan, khám phá các phương pháp giải quyết vấn đề cũng như những phạm vi và giới hạn của các nghiên cứu đó. Qua đó, ta có thể đánh giá tính khả thi của đề tài.

**Chương 3: Kiến thức nền tảng.** Trong chương này giới thiệu về dòng dữ liệu, và các tính chất của nó. Các phương pháp truy vấn và xử lý dữ liệu trên dòng dữ liệu. Giới thiệu về HyperLogLog và nguyên lý hoạt động của nó. Các kiến thức cơ bản về ước lượng số lượng phần tử trong dữ liệu lớn, và các khái niệm về đánh giá hiệu suất và độ chính xác trong đếm số phần tử 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.** Trong chương này sẽ tóm lại các ưu điểm và nhược điểm của mô hình và đưa ra các hướng nghiên cứu phát triển các kỹ thuật đếm số phần tử trong tương lai.

# 9 KẾT LUẬN

Việc giám sát và quản lý số lượng người dùng đóng vai trò quan trọng trong việc tối ưu hóa hiệu quả hoạt động, nâng cao trải nghiệm người dùng, hỗ trợ ra quyết định kinh doanh sáng suốt và đảm bảo an ninh mạng cho doanh nghiệp.

Phân bổ tài nguyên hợp lý: Đảm bảo hệ thống hoạt động ổn định, tránh quá tải, lãng phí tài nguyên, tối ưu hóa chi phí vận hành.

Nâng cao trải nghiệm người dùng: Giảm thiểu lỗi hệ thống, lag, giật, loading lâu, mang đến trải nghiệm mượt mà, thu hút và giữ chân khách hàng.

Phát hiện và khắc phục sự cố kịp thời: Nhận diện sớm các dấu hiệu bất thường, sự cố hệ thống, từ đó có biện pháp khắc phục nhanh chóng, hạn chế ảnh hưởng đến hoạt động kinh doanh.

Hiểu rõ hành vi người dùng: Phân tích dữ liệu truy cập, hành vi click chuột, sở thích, nhu cầu của người dùng để cá nhân hóa trải nghiệm, đề xuất sản phẩm/dịch vụ phù hợp, nâng cao hiệu quả marketing và dự báo xu hướng thị trường.

Hỗ trợ ra quyết định kinh doanh: Đánh giá hiệu quả chiến dịch marketing, điều chỉnh chiến lược phù hợp, dự báo nhu cầu thị trường, lập kế hoạch sản xuất, kinh doanh hiệu quả, tối ưu hóa nguồn lực, nâng cao lợi nhuận cho doanh nghiệp.

Sự bùng nổ của kỷ nguyên dữ liệu lớn đặt ra nhu cầu cấp thiết cho các kỹ thuật xử lý dữ liệu hiệu quả, chính xác. Đề tài "NGHIÊN CỨU PHÁT TRIỂN KỸ THUẬT ĐẾM SỐ PHẦN TỬ TRÊN DÒNG DỮ LIỆU" là một đóng góp quan trọng cho lĩnh vực khoa học máy tính. Các thuật toán mới được phát triển trong đề tài này giúp nâng cao hiệu quả và độ chính xác của việc xử lý dữ liệu, mang đến những đóng góp to lớn cho lĩnh vực khoa học máy tính và mở ra cánh cửa cho vô số ứng dụng thực tiễn.

# Tài liệu

[1] 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*. IEEE. 2010, pp. 1297–1303.

[2] 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*. Springer. 2003, pp. 605–617.

[3] Otmar Ertl. “ExaLogLog: Space-Efficient and Practical Approximate Distinct Counting up to the Exa-Scale”. In: *arXiv preprint arXiv:2402.13726* (2024).

[4] Philippe Flajolet et al. “Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm”. In: *Discrete mathematics & theoretical computer science* Proceedings (2007).

[5] 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*. 2013, pp. 683–692.