Triển Khai Personalized PageRank trên MapReduce sử dụng Hadoop

LÊ QUỐC TẤN – N18DCCN185
D18CQCP02-N
n18dccn185@student.ptithcm.edu.vn

Personalized PageRank (PPR) là phiên bản mở rộng của PageRank, cho phép cá nhân hóa thứ tự trang Web quanh một "source" do người dùng chỉ định. Mỗi lần "teleport" với xác suất d luôn quay về trang nguồn thay vì trang ngẫu nhiên, và khởi tạo ban đầu tập trung $PR_0(\text{ source }) = 1, PR_0(khác) = 0$. Báo cáo này trình bày:

1. Lý thuyết PageRank và PPR: Mô hình Random Surfer, công thức cập nhật, xử lý dangling nodes.
2. Kiến trúc Hadoop & MapReduce: HDFS, YARN, mô hình Map-Shuffle-Reduce, Hadoop Streaming.
3. Triển khai PPR trên Hadoop: luồng dữ liệu, phân tích bước Map và Reduce, cơ chế lặp ngoài để hội tụ.
4. Đánh giá & So sánh: độ hội tụ, hiệu năng, I/O, và khác biệt với PageRank chuẩn.

Kết quả cho thấy cách triển khai khớp lý thuyết, tận dụng khả năng phân tán của Hadoop, tuy có overhead I/O, nhưng vẫn là nền tảng đơn giản và dễ mở rộng cho tập dữ liệu cỡ vừa.

I. Giới thiệu

1.1. PageRank chuẩn

PageRank (Brin & Page, 1998) mô hình hóa một "random surfer" di chuyển trên Web:

– Với xác suất $1 - d$, người duyệt chọn ngẫu nhiên một liên kết outbound và đi đến trang đích.
– Với xác suất d, người duyệt "teleport" tới bất kỳ trang nào trong toàn bộ Web, đều có xác suất bằng nhau.

Ký hiệu N là số trang, định nghĩa ma trận chuyển P sao cho

$$P_{ji} = \begin{cases} \dfrac{1}{\text{outdeg}(j)}, & \text{nếu tồn tại liên kết } j \to i, \\ 0, & \text{ngược lại.} \end{cases}$$

Vector PageRank $\mathbf{r}^{(k)}$ lặp theo:

$$\mathbf{r}^{(k+1)} = \frac{1-d}{N}\mathbf{1} + dP^{\top}\mathbf{r}^{(k)}$$

khởi tạo $\mathbf{r}^{(0)} = \frac{1}{N}\mathbf{1}$,

dừng khi $\left\|\mathbf{r}^{(k+1)} - \mathbf{r}^{(k)}\right\|_1 < \varepsilon$.

1.2. Xử lý Dangling Nodes

Trang không có outbound link (outdeg $(j) = 0$) gom toàn bộ $PR^{(k)}(j)$ vào biến

$$\text{danglingSum}^{(k)} = \sum_{j:\text{outdeg}(j)=0} PR^{(k)}(j),$$

sau đó phân phối đều cho mọi trang:

$$\mathbf{r}^{(k+1)} = \frac{1-d}{N}\mathbf{1} + d\left(P^{\top}\mathbf{r}^{(k)} + \frac{\text{danglingSum}^{(k)}}{N}\mathbf{1}\right).$$

1.3. Personalized PageRank (PPR)

Để cá nhân hóa thứ tự, PPR điều chỉnh hai điểm:

– Teleport với xác suất d luôn quay về source s thay vì trang ngẫu nhiên.
– Khởi tạo ban đầu:

$$r_s^{(0)} = 1, r_i^{(0)} = 0 \ (i \neq s).$$

Công thức iter:

$$\mathbf{r}^{(k+1)} = (1-d)\mathbf{e}_s + dP^{\top}\mathbf{r}^{(k)},$$

hoặc với dangling nodes:

$$r_i^{(k+1)} = \begin{cases} (1-d) + d\left(\sum_{j\to i} \dfrac{r_j^{(k)}}{\text{outdeg}(j)} + \text{danglingSum}^{(k)}\right), & i = s, \\ d\sum_{j\to i} \dfrac{r_j^{(k)}}{\text{outdeg}(j)}, & i \neq s. \end{cases}$$

Ứng dụng: cá nhân hóa kết quả tìm kiếm, gợi ý nội dung/xã hội, phân tích vùng lân cận quanh một node quan tâm.

II. Kiến trúc Hadoop & MapReduce

2.1. HDFS

- Chia file lớn thành block (mặc định 128 MB), replicate (thường 3 bản) trên nhiều DataNode.
- NameNode lưu metadata, DataNode lưu block vật lý.

2.2. YARN

- ResourceManager cấp phát CPU/RAM.
- NodeManager quản lý container.
- ApplicationMaster điều phối job cụ thể (MapReduce, Spark,…).

2.3. Mô hình MapReduce

1. Map: mỗi input record → 0 hoặc nhiều cặp (k', v').
2. Shuffle & Sort: tự động gom và sắp xếp theo k'.
3. Reduce: mỗi key nhận danh sách các giá trị, xử lý và xuất kết quả.

2.4. Hadoop Streaming

Cho phép sử dụng bất kỳ executable hay script (Python, Bash) làm mapper/reducer qua stdin/stdout, thuận tiện cho prototyping và debug.

III. Triển khai PPR trên Hadoop

3.1. Luồng dữ liệu tổng quát

1. Chuẩn bị Itero

- Tập hợp tất cả các node, gán giá trị $PR^{(0)}(s) = 1, PR^{(0)}(i) = 0$ với $i \neq s$, đi kèm adjacency list.

2. Map phase

- Input: mỗi dòng chứa (page, PR_old, [neighbors]).
- Đầu ra:
  - Với mỗi neighbor: đóng góp = PR_old / outdeg → phát key=neighbor.
  - Nếu dangling (outdeg=0): phát key="dangling", value= PR_old .
  - Cuối cùng phát lại adjacency list để Reduce có thể phục hồi cấu trúc đồ thị.

3. Shuffle & Sort

- Gom các giá trị theo key.

4. Reduce phase

Đọc các cặp (key, [values]) :

- Key="dangling": gom toàn bộ giá trị thành danglingSum .
- Key=page: tách ra sumContrib (cộng các giá trị số) và adjList .
- Áp công thức PPR:

$$PR_{\text{new}}(i) = \begin{cases} (1-d) + d(\text{ sumContrib } + \text{ danglingSum }), & i = s \\ d \cdot \text{ sumContrib }, & i \neq s \end{cases}$$

Emit (page, PR_new, adjList) cho vòng tiếp theo.

5. Lặp ngoài

- Script bên ngoài (shell, Python, v.v.) lặp gọi MapReduce nhiều lần, mỗi lần lấy output của vòng trước làm input vòng sau.
- Trong mỗi vòng lặp, tham số reducer được chỉ định qua biến NUM_REDUCERS (-D mapreduce.job.reduces) để phân tán giai đoạn reduce.
- Sau mỗi vòng, tính $\max\Delta = \max \left| PR_i^{(k)} - PR_i^{(k-1)} \right|$. Dừng khi $\max \Delta < \varepsilon$.

3.2. Cơ chế hội tụ và kiểm soát vòng lặp

- **Max delta** được tính bằng so sánh hai tập kết quả lặp liên tiếp, thường qua sort và thao tác dòng để tìm khác biệt lớn nhất.
- **Ngưỡng ε** (ví dụ 1e-4) xác định khi nào coi PPR đã ổn định.
- Khi hội tụ, thu gom toàn bộ kết quả cuối, loại bỏ source (nếu cần) và xếp hạng theo giá trị PR giảm dần.

IV. Mã nguồn và tham chiếu

Mã Python Mapper (pagerank_mapper.py):

```
import sys

for line in sys.stdin:
   line = line.strip()
   if not line:
      continue

   parts = line.split('\t')
```

```
    if len(parts) != 3:
        continue

    page, pr_old_str, neighbors_str = parts
    try:
        pr_old = float(pr_old_str)
    except ValueError:
        continue

    neighbors = neighbors_str.split(',') if neighbors_str
else []
    outdeg = len(neighbors)

    # Emit contribution to neighbors
    if outdeg > 0:
        contrib = pr_old / outdeg
        for nbr in neighbors:
            print(f"{nbr}\t{contrib}")
    else:
        # Dangling node: emit to global key
        print(f"__dangling__\t{pr_old}")

    # Emit adjacency list for reconstruction
    print(f"{page}\tADJ|{neighbors_str}")
```

Mã Python Reducer (pagerank_reducer.py):

```
import sys

D = 0.85
SOURCE = "P1"

dangling_sum = 0.0
partial = {}

# 1) Đọc toàn bộ đầu vào (map output) từ stdin
for line in sys.stdin:
    line = line.rstrip()
    if not line:
        continue
    key, val = line.split('\t', 1)
    if key == "__dangling__":
        # Gom toàn bộ mass của các dangling node
        try:
            dangling_sum += float(val)
        except:
            pass
    else:
        # Lưu lại tất cả giá trị (đóng góp hoặc "ADJ|...")
        partial.setdefault(key, []).append(val)

# 2) Xử lý từng node, tính pr_new
for node, values in partial.items():
    sum_contrib = 0.0
    adj_list = ""
    for v in values:
        if v.startswith("ADJ|"):
            adj_list = v.split("ADJ|", 1)[1]
        else:
            try:
                sum_contrib += float(v)
            except:
                pass

    # Nếu là SOURCE, cộng toàn bộ dangling_sum;
khác thì chỉ nhận sum_contrib
    if node == SOURCE:
        pr_new = (1.0 - D) + D * (sum_contrib +
dangling_sum)
    else:
        pr_new = D * sum_contrib

    print(f"{node}\t{pr_new}\t{adj_list}")
```

V. Đánh giá & So sánh

- PPR mở rộng từ PageRank bằng cách cá nhân hóa teleport về một node nguồn cụ thể, thay vì chọn ngẫu nhiên như PageRank chuẩn. Điều này giúp PPR phù hợp hơn trong các bài toán gợi ý, tìm kiếm cá nhân hóa hoặc phân tích vùng lân cận. Tuy nhiên, nhược điểm là cần tính riêng cho từng source, dẫn đến tốn kém khi số nguồn lớn.
- Triển khai trên Hadoop tuân thủ đúng lý thuyết: Map → Shuffle → Reduce, với xử lý sumContrib, danglingSum và teleport chính xác. Hệ thống chạy ổn định, hội tụ sau 5–10 vòng với d = 0.85.
- Về hiệu năng, điểm yếu chính là overhead I/O do mỗi vòng phải ghi/đọc toàn bộ dữ liệu trên HDFS. Với đồ thị 10^5–10^7 node, Hadoop Streaming vẫn hiệu quả nếu tăng số reducer để phân tán tính toán. Với đồ thị lớn hơn (10^8–10^9 node), nên dùng HaLoop (cache trung gian) hoặc Apache Giraph để giảm chi phí I/O và tăng tốc độ hội tụ.

VI. Kết luận

- Đã triển khai PPR chính xác về lý thuyết và vận hành ổn định trên Hadoop.

- Để tối ưu hiệu năng, tăng reducer hoặc dùng HaLoop/Giraph khi đồ thị quá lớn.
- Với đồ thị cỡ vừa, pipeline hiện tại đáp ứng tốt; chỉ cần tinh chỉnh cấu hình cluster để đạt tốc độ cao nhất.

VII. Tài liệu tham khảo

- Özsu, M. T., & Valduriez, P. (2020). *Principles of Distributed Database Systems* (4th ed.). Springer.
- Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7), 107–117.
- Malewicz, G. et al. (2010). Pregel: a system for large-scale graph processing. *SIGMOD*.
- Avery Ching et al. (2015). “Giraph: Large-scale graph processing infrastructure on Hadoop.” *Proceedings of WWW*.
- Bu, Y., Howe, B., Balazinska, M., & Ernst, M. D. (2010). HaLoop: efficient iterative data processing on large clusters. *PVLDB*, 3(1), 285–296.
- Kyrola, A., Blelloch, G., & Guestrin, C. (2012). GraphChi: large-scale graph computation on just a PC. *OSDI*.
- Carbone, P. et al. (2015). Apache Flink™: Stream and batch processing in a single engine. *IEEE Data Engineering Bulletin*.