

Báo cáo về Chord và ứng dụng

1. Mô tả cụ thể về thuật toán Chord

1.1. Giới thiệu

- ❖ Chord là một thuật toán Distributed Hash Table (Bảng băm phân tán), là hệ thống phân tán không tập trung, cung cấp dịch vụ tra cứu dữ liệu dựa trên các cặp khóa – giá trị.
- ❖ Mục tiêu chính: xác định node nào trong mạng chịu trách nhiệm lưu trữ một khóa (key) nhất định.
 - Đảm bảo khả năng mở rộng tốt khi mạng có rất nhiều node.
- ❖ Độ phức tạp của tra cứu: $O(\log N)$ với N là số lượng Node trong hệ thống.
- ❖ Thuật toán này phân tán dữ liệu vào nhiều node theo một vòng tròn định danh có kích thước 2^m .

1.2 Nguyên tắc hoạt động

- Toàn bộ các node và key được ánh xạ vào một không gian định danh từ 0 đến $2^m - 1$, m là số bit định danh.
- Mỗi key được gán cho node đầu tiên có $ID \geq key$ theo chiều kim đồng hồ.
- Nếu không có node nào có $ID \geq key$, thì key thuộc về node nhỏ nhất (quay vòng).
- Finger table: để tra cứu nhanh, mỗi node duy trì một finger table gồm m bản ghi. Tại node n thì finger table được tính như sau:

$$FTp[i] = \text{succ}(p + 2^{(i-1)})$$

1.3. Tìm kiếm key (Look up)

Để tìm node chịu trách nhiệm cho một key k :

1. Hash key \rightarrow lấy ID.
2. Kiểm tra finger table của node hiện tại.
3. Nếu k nằm trong $(n, \text{successor}]$, trả về successor.
4. Ngược lại, chọn node trong finger table **gần key nhất nhưng nhỏ hơn key** \rightarrow chuyển tiếp truy vấn đến node đó.
5. Lặp lại cho đến khi tìm được node đích.

2. Minh chứng kết quả test case

- Cho $m = 4$ nên id sẽ nằm trong $[0, 15]$.
- Cấu hình cho danh sách node là $[2, 6, 8, 13]$.
- Xong rồi tự đặt key để mô phỏng chương trình:

```
keys = {  
    "Dung": 9,  
    "Duong": 12,  
    "Nhưng": 6,  
    "Mai": 3,  
    "Long": 14  
}
```

Case 1: Dung, có ID = 9, node đầu tiên ≥ 9 là node 13 (theo chiều kim đồng hồ) nên node chịu trách nhiệm là 13.

Case 2: Duong, có ID = 12, node đầu tiên ≥ 9 là node 13 (theo chiều kim đồng hồ) nên node chịu trách nhiệm là 13.

Case 3:Nhung, có ID = 6, node đầu tiên ≥ 6 là node 6 (theo chiều kim đồng hồ) nên node chịu trách nhiệm là 6.

Case 4:Mai, có ID = 3, node đầu tiên ≥ 3 là node 6 (theo chiều kim đồng hồ) nên node chịu trách nhiệm là 6.

Case 5: Long, có ID = 14, node đầu tiên ≥ 14 là node 2 (theo chiều kim đồng hồ) nên node chịu trách nhiệm là 2.

Kết quả test thực tế

```
=== Test Lookup Key ===
Key Huy (ID=9) -> Node 13
Key Duong (ID=12) -> Node 13
Key An (ID=6) -> Node 6
Key Bach (ID=3) -> Node 6
Key Tuan (ID=14) -> Node 2
```

- Từ đó ta thấy, actual output trùng với đầu ra mà chúng ta mong đợi.

3. Code thực nghiệm

- Hàm `find_successor`: tìm successor cho một key hoặc một start.
- Hàm `build_finger_table`: Xây dựng finger table cho tất cả các node.
- Main: Dùng mô phỏng test case.

```
- def find_successor(key, sorted_nodes):
-     for node in sorted_nodes:
-         if node >= key:
-             return node
-     return sorted_nodes[0]
-
- def build_finger_table(nodes, m):
-     finger_tables = {}
-     sorted_nodes = sorted(nodes)
-     for node in sorted_nodes:
-         finger_tables[node] = []
-         for i in range(m):
-             start = (node + 2 ** i) % (2 ** m)
-             successor = find_successor(start, sorted_nodes)
```

```

-         finger_tables[node].append((start, successor))
-     return finger_tables
-
- if __name__ == "__main__":
-
-     m = 4
-     nodes = [2, 6, 8, 13]
-     sorted_nodes = sorted(nodes)
-
-     finger_tables = build_finger_table(nodes, m)
-
-     print("=== Finger Table ===")
-     for node, table in finger_tables.items():
-         print(f"\nNode {node}:")
-         for i, (start, successor) in enumerate(table):
-             print(f"  Finger {i+1}: start={start}, successor={successor}")
-
-     keys = {
-         "Dung": 9,
-         "Duong": 12,
-         "Nhưng": 6,
-         "Mai": 3,
-         "Long": 14
-     }
-
-     print("\n=== Test Lookup Key ===")
-     for key, key_id in keys.items():
-         successor = find_successor(key_id, sorted_nodes)
-         print(f"Key {key} (ID={key_id}) -> Node {successor}")
-

```

4. Kết luận

- Thuật toán Chord giúp xác định node chịu trách nhiệm quản lý dữ liệu của từng người dùng nhanh chóng.
- Nhờ bảng finger table, tra cứu không cần quét toàn bộ nodes, giảm độ phức tạp xuống $O(\log N)$.
- Ứng dụng cụ thể: Hệ thống phân tán quản lý người dùng. Mỗi node chịu trách nhiệm lưu thông tin người dùng, thuận tiện cho việc mở rộng và chịu lỗi.

Link github: <https://github.com/DoanHongDuong/Chord>.