## khung doi

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**BỘ MÔN HỆ THỐNG THÔNG TIN**



**BÁO CÁO BÀI TẬP**

**MÔN: Tính Toán Phân Tán**

# Đề BÀI : giải mã mật khẩu md5 sử dụng tính toán trên mpi trên rocks cluster

**SV thực hiện:**

1. Cao Thanh Tùng : 20145075

**Hà Nội, Tháng 04 năm 2018**

**Mục lục**

[1](#_Toc510690061)

[Đề BÀI : giải mã mật khẩu md5 sử dụng tính toán trên mpi trên rocks cluster 1](#_Toc510690062)

[1. Giới thiệu bài toán 3](#_Toc510690063)

[2. Phương pháp 3](#_Toc510690064)

[3. Cài đặt 4](#_Toc510690065)

[4. Thử nghiệm 6](#_Toc510690066)

[5. kết luận 8](#_Toc510690067)

# Giới thiệu bài toán

Giả thiết của bài toán là đã có đoạn mã được sinh ra bởi MD5 và cho biết độ dài của mật khẩu ban đầu cũng như bảng chữ cái mà mật khẩu đó được sinh ra.

Yêu cầu của bài toán là cần thám mã với đoạn mã được hash bởi MD5 đó ta thực hiện thám mã để có thể thu được mật khẩu trước khi được mã hóa.

VD : ta đã có mã MD5 của mật khẩu là “6ae4889bcc10a454ff0891e145e8c40c” và giả sử mật khẩu chỉ sử dụng kí tự trong bảng chữ tiếng Anh in thường, mật khẩu với độ dài là 6 kí tự

Yêu cầu : là tìm ra được mật khẩu ban đầu. Với trường hợp này, ta cần thu được mật khẩu là “tungct”.

# Phương pháp

Sử dụng kiểu tấn công vét cạn (Brute force)

Brute force sẽ vét cạn các trường hợp có thể có của mật khẩu, rồi sinh ra đoạn mã theo MD5, từ đó ta so sánh đoạn mã của mật khẩu với đoạn mã đang xét, nếu hai đoạn mã trùng nhau thì ta thu đã “tìm” được mật khẩu. Nếu đoạn mã đang xét khác đoạn mã của mật khẩu thì ta tiếp tục thử với trường hợp tiếp theo, cứ như vậy cho đến khi tìm được mật khẩu.

Ví dụ : Mật khẩu “tungct” là mật khẩu mà ta cần phải đoán ra

Giả sử ta đã thu được đoạn mã sinh ra theo MD5 của mật khẩu “tungct” là “6ae4889bcc10a454ff0891e145e8c40c”, và độ dài mật khẩu là 6 và giả sử mật khẩu chỉ là các chữ cái tiếng anh in thường (bảng chữ cái 26 kí tự)

Phương pháp :

Với độ dài mật khẩu là 6, thì tất cả các trường hợp mật khẩu ta phải vét cạn là 26 ^ 6 trường hợp, từ “aaaaaa”, “aaaaab”, ..., “zzzzzy”, “zzzzzz”

Với từng trường hợp mật khẩu đang xét, ta sinh mã MD5 của nó, ví dụ khi xét đến “tungcs” ta có mã MD5 tương ứng là “e4a3c2a082e12555f8cf1f179cddf8ef”, ta đem so sánh với mã MD5 của mật khẩu cần tìm, ta thấy vẫn chưa trùng nhau, ta xét tiếp mật khẩu tiếp theo là “tungct” với mã MD5 tương ứng là “6ae4889bcc10a454ff0891e145e8c40c”, khi so sánh mã này với mã MD5 của mật khẩu gốc, ta thấy hoàn toàn trùng khớp. Do vậy ta dã “tìm” được mật khẩu là “tungct”.

# Cài đặt

Sử dụng phương pháp vét cạn như trên miêu tả, tuy nhiên để tăng tốc độ tìm mật khẩu, ta cần lập trình song song để có thể cùng lúc thử được các trường hợp khác nhau của mật khẩu để thu ngắn thời gian. Cụ thể hơn, ta thiết kế chương trình trên MPI để nâng cao hiệu quả .

Ý tưởng chung là thay vì vét cạn trên 1 tiến trình của 1 máy, ta sử dụng nhiều máy, nhiều tiến trình và chia việc thử các trường hợp cho các tiến trình chạy trên các node để cùng thực hiện cho đến khi 1 tiến trình bất kỳ tìm được mật khẩu. Ta cần chia đều số mật khẩu cần thử cho các tiến trình.

Thiết kế trên MPI :

Để thử nghiệm, tiến hành thử nghiệm trên 2 cách cài đặt :

* Cài đặt 1 : Coi các node có nhiệm vụ như nhau, đề thực hiện công việc tính toán để tìm ra mật khẩu.
* Cài đặt 2 : Coi node frontend với rank 0 là node master để quản lý, phân chia, gửi công việc cho các node khác để tính toán tìm ra mật khẩu, sau đó đợi và nhận kết quả gửi về từ các node.

Thiết kế với từng cách cài đặt :

Cài đặt 1 : Sử dụng rank và tổng số tiền trình để thực hiện việc phân chia :

* Ta coi mỗi mật khẩu cần thử tương ứng với 1 số nguyên (chính là số thứ tự mật khẩu trong danh sách cần thử), qua đó việc phân chia các mật khẩu sẽ chính là phân chia khoảng thứ tự của mật khẩu đó cho các tiến trình trên các node với rank tương ứng, sao cho số lượng các mật khẩu cần thử là gần như đều nhau.

VD : với danh sách các mật khẩu 1 kí tự cần thử từ “a” , “b”, ... “z”, thì “a” sẽ ứng với 0, “b” ứng với 1,... “z” ứng với 25

Với việc chạy trên 2 tiến trình trên 2 máy, ta chia được như sau :

Tiến trình với rank 0 : thử các trường hợp từ 0-12 tương ứng với từ “a” đến “m”

Tiến trình với rank 1 : thử các trường hợp từ 13-25 tương ứng từ “n” đến “z”

* Với mỗi tiến trình với các khoảng mật khẩu cần tìm, sẽ thực hiện phép thử từ mật khẩu đầu tiên đến cuối cùng trong khoảng đó, cho đến khi tìm được mật khẩu, khi đó MPI sẽ dừng lại.

Cài đặt 2 : Vẫn dùng ý tưởng sử dụng rank và tổng số tiền trình để thực hiện việc phân chia, tuy nhiên có 1 chút khác biệt do với node frontend với rank 0 sẽ làm nhiệm vụ điều phối việc phân chia và nhận kết quả :

* Ta coi mỗi mật khẩu cần thử tương ứng với 1 số nguyên (chính là số thứ tự mật khẩu trong danh sách cần thử), qua đó việc phân chia các mật khẩu sẽ chính là phân chia khoảng thứ tự của mật khẩu đó cho các tiến trình trên các node với rank tương ứng (trừ node frontend với rank 0), sao cho số lượng các mật khẩu cần thử là gần như đều nhau. Việc phân chia này được tính toán trên node với rank 0 – node frontend, sau đó, node frontend sẽ thực hiện việc gửi giá trị khoảng giá trị đến các node với rank tương ứng mà nó tính toán. Các node tính toán này sau khi nhận công việc từ frontend, nó tiến hành tìm kiếm mật khẩu, khi tìm thấy hoặc tính toán xong, các node này sẽ gửi trả lại kết quả đến node frontend. Node frontend nhận kết quả từ các node, sẽ biết được thông tin cụ thể như mật khẩu tìm thấy là gì và node nào tìm ra mật khẩu này, qua đó đưa ra kết quả cuối cùng.

VD : với danh sách các mật khẩu 1 kí tự cần thử từ “a” , “b”, ... “z”, thì “a” sẽ ứng với 0, “b” ứng với 1,... “z” ứng với 25

* Việc phân chia mật khẩu được thực hiện

Với việc chạy trên 3 máy, ta chia được như sau :

Với rank 0 – node frontend : tính toán được rằng có 26 khả năng từ 0-25 của mật khẩu, và biết được có 2 node phục vụ cho việc tính toán với rank tương ứng là 1 và 2. Do đó rank 0 sẽ gửi thông tin đến từng node này : rank 1 sẽ tính toán trường hợp từ 0-12, rank 2 sẽ tính toán trường hợp từ 13-25 và đợi kết quả gửi về từ 2 rank này.

Tiến trình với rank 1 : thử các trường hợp từ 0-12 tương ứng với từ “a” đến “m”, nếu tìm được mật khẩu hoặc chương trình kết thúc, gửi trả về kết quả cho rank 0

Tiến trình với rank 2 : thử các trường hợp từ 13-25 tương ứng từ “n” đến “z”, nếu tìm được mật khẩu hoặc chương trình kết thúc, gửi trả về kết quả cho rank 0

Cuối cùng, rank 0 sẽ thu được kết quả từ 2 rank này, đưa ra kết quả cuối cùng.

# Thử nghiệm

Thực hiện thử nghiệm trên máy ảo Virtualbox cài đặt Rocks Cluster 6.2

* Rocks-frontend :
  + Số lượng : 1
  + Memory : 1 GB
  + Disk : 30 GB
* Rocks-compute :
  + Số lượng : 2
  + Memory : 1 GB
  + Disk : 30 GB
* Bảng ký tự của mật khẩu là bảng chữ cái viết thường tiếng Anh 26 kí tự
* Mật khẩu xét lần lượt là “zzzz”, “zzzzzz” với độ dài tương ứng lần lượt là 4, 6 kí tự để việc tìm kiếm mật khẩu phải vét tất cả các trường hợp ( mà không phụ thuộc vào vị trí của mật khẩu đó khi chia cho tiến trình tính toán . VD : nếu xét mật khẩu là “aac” mà tiến trình với rank 0 được chia tìm kiếm từ mật khẩu là “aaa” thì việc tìm kiếm sẽ kết thúc sớm )
* Thử nghiệm trên các trường hợp :
  + 1 tiến trình / 1 máy
  + 4 tiến trình / 1 máy
  + 4 tiến trình / 2 máy
  + 6 tiến trình / 2 máy
  + 6 tiến trình / 3 máy
  + 12 tiến trình / 3 máy

Kết quả thu được :

+ Cách cài đặt 1 : Các node đều có vai trò như nhau, đề thực hiện tính toán :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Số tiến trình | Số máy | Mật khẩu | Thời gian tìm ra (s) | Số mật khẩu / 1s |
| 1 | 1 | zzzz | 0.800000 | 571220 |
| 4 | 1 | zzzz | 0.210000 | 2176076 |
| 4 | 2 | zzzz | 0.250000 | 1827904 |
| 6 | 2 | zzzz | 0.210000 | 2176076 |
| 6 | 3 | zzzz | 0.250000 | 1827904 |
| 12 | 3 | zzzz | 0.130000 | 3515200 |
| 1 | 1 | zzzzzz | 577.960022 | 534493 |
| 4 | 1 | zzzzzz | 144.490005 | 2137973 |
| 4 | 2 | zzzzzz | 162.919998 | 1896119 |
| 6 | 2 | zzzzzz | 110.220001 | 27532598 |
| 6 | 3 | zzzzzz | 165.669998 | 1864645 |
| 12 | 3 | zzzzzz | 99.070000 | 3118156 |

+ Cách cài đặt 2 : node frontend – rank 0 sẽ làm node master, thực hiện phân chia công việc cho các node khác, các node khác thực hiện nhiệm vụ tính toán (Số tiến trình và số máy không tính rank 0) :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Số tiến trình | Số máy | Mật khẩu | Thời gian tìm ra (s) | Số mật khẩu / 1s |
| 1 | 1 | zzzz | 0.910000 | 502171 |
| 4 | 1 | zzzz | 0.360000 | 1269377 |
| 4 | 2 | zzzz | 0.320000 | 1428050 |
| 6 | 2 | zzzz | 0.210000 | 2176076 |
| 1 | 1 | zzzzzz | 691.400024 | 446797 |
| 4 | 1 | zzzzzz | 221.979996 | 1391638 |
| 4 | 2 | zzzzzz | 227.759995 | 1356321 |
| 6 | 2 | zzzzzz | 150.130005 | 2057655 |

* Thử nghiệm bổ sung với cách cài đặt thứ 1 trường hợp mật khẩu xét lần lượt là “tung”, “tungct” với độ dài tương ứng lần lượt là 4, 6 kí tự
* Kết quả thu được :

|  |  |  |  |
| --- | --- | --- | --- |
| Số tiến trình | Số máy | Mật khẩu | Thời gian tìm ra (s) |
| 1 | 1 | tung | 0.640000 |
| 4 | 1 | tung | 0.010000 |
| 4 | 2 | tung | 0.020000 |
| 6 | 2 | tung | 0.090000 |
| 6 | 3 | tung | 0.150000 |
| 12 | 3 | tung | 0.020000 |
| 1 | 1 | tungct | 430.200012 |
| 4 | 1 | tungct | 6.490000 |
| 4 | 2 | tungct | 7.160000 |
| 6 | 2 | tungct | 63.529999 |
| 6 | 3 | tungct | 94.230003 |
| 12 | 3 | tungct | 11.500000 |

# kết luận

* Việc tìm mật khẩu bằng tính toán song song cụ thể hơn trong bài này là sử dụng MPI là rất hiệu quả để thực hiện việc dò mật khẩu một cách nhanh chóng
* Tuy nhiên, với vấn đề về cấu hình của từng node, CPU hay Memory và cấu hình của máy tính cài đặt rock-cluster khi tính toán, chi phí thời gian truyền thông giữa các node thì việc tính toán có thể bị chậm chễ hơn so với sử dụng 1 node
* Kết quả thời gian thực hiện tìm ra mật khẩu phụ thuộc vào mật khẩu tương ứng với vị trí bao nhiêu đối với node tính toán (VD : khi tìm mật khẩu có 1 kí tự, tương ứng với 26 trường hợp 0-25, 2 máy tính toán, máy 1 là 0-12, máy 2 là 13-25, xét trường hợp mật khẩu tương ứng với kí tự c – với số thứ tự là 2, thì kết quả tìm ra sẽ là nhanh hơn so với trường hợp mật khẩu là m – tương ứng với số thứ tự 12 (máy 1 sẽ phải vét hết các trường hợp) .