ISSN 2354-0575

PHÂN TÍCH HIỆU NĂNG CỦA KIẾN TRÚC TÍNH TOÁN SONG SONG TRÊN NỀN HADOOP FRAMEWORK

Nguyễn Minh Quý, Hồ Khánh Lâm, Nguyễn Xuân Trường, Nguyễn Đình Hân, Nguyễn Văn Hậu, Đỗ Anh Tuấn, Phạm Quốc Hùng

Trường Đại học Sư phạm Kỹ thuật Hưng Yên

Ngày tòa soạn nhận được bài báo: 08/04/2019
Ngày phản biện đánh giá và sửa chữa: 07/05/2019
Ngày bài báo được duyệt đăng: 16/05/2019

Tóm tắt:

Kiến trúc tính toán song song dựa trên hệ thống cụm máy tính sử dụng Hadoop framework trong vài năm trở lại đây nhận được sự quan tâm rất lớn vì khả năng tính toán và mở rộng của hệ thống khá thuận lợi và dễ dàng, đặc biệt là trong xử lý dữ liệu cực lớn lên tới hàng Tera-byte hay thậm chí là Peta-byte. Tuy nhiên, hiệu năng của kiến trúc này phụ thuộc vào nhiều yếu tố như cài đặt thuật toán, tối ưu hóa câu lệnh HiveQL, kiến trúc cluster, v.v... Bài báo này sẽ đề xuất giải pháp sử dụng kiến trúc tính toán song song Hive trên nền Hadoop cho bài toán xử lý luồng gói tin trong mạng nhằm phát hiện sự có mặt của Virus đồng thời phân tích và đưa ra một số biện pháp cải tiến hiệu năng cho hệ thống tính toán.

Từ khóa: *Hadoop, big data, parallel processing, performance analysis, tính toán song song.*

I. GIỚI THIỆU

Trong lĩnh vực phân tích các thông tin di chuyển trong mạng (Network Traffic), phương pháp giám sát dựa trên các luồng (Flow) là một trong những phương pháp được sử dụng rất rộng rãi. Việc phân tích luồng có thể giúp chúng ta: phát hiện được sự có mặt của Virus, tấn công DDoS, phát hiện mất cân bằng tải, mức độ sử dụng, v.v… Lý do người ta sử dụng phương pháp này đó là bởi vì nó không đi vào phân tích nội dung của gói tin (Packet Content) mà đi vào phân tích các thông tin công khai liên quan đến luồng gói tin đó, ví dụ như: Địa chỉ nguồn, địa chỉ đích, cổng nguồn, cổng đích, loại giao thức, kích thước gói tin, v.v… Các thông tin này có thể bắt (capture) bởi các router hoặc các công cụ phần mềm theo dõi chuyên dụng. Một số công cụ bắt gói tin được dùng phổ biến như [1,2,3].

Tuy nhiên, các công cụ phân tích theo phương pháp truyền thống gặp một trở ngại rất lớn đó là lưu lượng gói tin trong mạng sẽ rất lớn, có thể lớn tới hàng Giga-byte hoặc thậm chí là hàng Tera-byte, Peta-byte tùy theo quy mô và mức độ sử dụng của mạng. Ví dụ như [4], trung bình một ngày, dung lượng thông tin bắt về vào khoảng 40 GB. Với dung lượng lớn này, các công cụ hiện hành gần như là không thể phân tích được trên một trạm xử lý đơn lẻ.

Gần đây, một hướng tiếp cận mới trong việc xử lý dữ liệu lớn (big data) đó là song song và phân tán hóa quá trình xử lý và lưu trữ sử dụng Hadoop Framework. Hadoop được phát triển bởi Google, Yahoo, Facebook,… và đã cho những kết quả rất khả quan. Theo kết quả đánh giá năm 2013,… Hadoop đã về nhất khi sắp xếp 102.5 TB mất 4.328s với 2100 nodes [5].

Trong bài báo này, chúng tôi sẽ đưa ra mô hình xử lý luồng gói tin trên nền tảng Hadoop Framework ở phần II. Phần III sẽ đề cập một nền tảng khác là Hive Framework [6] được phát triển trên nền Hadoop để làm đơn giản hóa tương tác giữa nhà phát triển và Hadoop, sử dụng các câu lệnh tương tác giống như truy vấn SQL quen thuộc. Phần IV sẽ đề xuất một cách làm tăng hiệu quả xử lý khi kết nối các bảng dữ liệu trong Hive bằng các hàm tự định nghĩa. Phần V sẽ tiến hành thử nghiệm và đánh giá đề xuất được đưa ra ở phần IV chứng minh tính hiệu quả về hiệu năng của hệ thống

II. XÂY DỰNG MÔ HÌNH XỬ LÝ LUỒNG GÓI TIN

1. Luồng gói tin (Network Flow)

Trong môi trường mạng, các trạm trao đổi với nhau bằng cách gửi các gói tin, mỗi gói tin ngoài nội dung bên trong thì luôn kèm theo các thông tin định tuyến như địa chỉ nguồn, địa chỉ đích, cổng nguồn, cổng đích, kích thước gói tin, kiểu gói tin, thời gian, loại giao thức được sử dụng, v.v… Các thông tin này hoàn toàn có thể bắt được bằng các phần mềm giám sát mạng như Wireshark [6], TcpDump,… hoặc các thiết bị phần cứng chuyên dụng [7] hoặc bằng chính các router Cisco đời mới có tính năng gọi là "Flow Export" [8].

2. Mô hình xử lý

Dưới đây là mô hình đề xuất cho việc bắt và xử lý luồng gói tin, kết hợp giữa Router và Hadoop

framework.

Tại đây, các gói tin đi qua sẽ được Router bắt lại và Export tới địa chỉ đích đã được cấu hình trước.

Các luồng gói tin này sẽ được lưu và xử lý trong hệ thống Hadoop. Kết quả xử lý của hệ thống Hadoop được kết xuất ra một file text.

3. Hadoop Framework

Hadoop [9] là một hệ thống cho phép lưu trữ dữ liệu trên hệ thống file phân tán HDFS (Hadoop Distrbuted File System) tại các nút (Node) và xử lý dữ liệu này thông qua một framework là MapReduce. MapReduce có hai hàm chính là hàm Map (Mapping) và hàm Reduce. Hai hàm này sẽ được người dùng viết lại tùy thuộc vào bài toán cụ thể. Tuy nhiên, việc tùy biến sử dụng 2 hàm này không phải lúc nào cũng dễ dàng. Vì vậy, để thuận tiện, một Framework khác là Hive [10] được phát triển đóng vai trò trung gian giữa nhà phát triển và Hadoop nhằm đơn giản hóa trong việc viết hai hàm Map và Reduce. Nhà phát triển khi đó có thể khai thác dễ dàng hệ thống Hadoop thông qua các câu lệnh HiveQL với cú pháp tương tự ngôn ngữ truy vấn SQL.

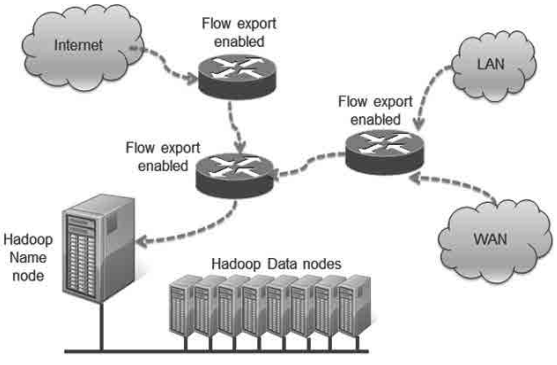
III. PHÂN TÍCH LUỒNG GÓI TIN VÀ PHÁT HIỆN VIRUS VỚI HIVE FRAMEWORK

1. Phát hiện sự có mặt của Virus thông qua phân tích luồng gói tin

Nhờ vào việc phân tích thông tin của các luồng gói tin, người ta hoàn toàn có thể đưa ra được các thông tin rất có ích, như: Phát hiện sự có mặt của Virus, phát hiện các cuộc tấn công DDoS, mức độ tải trên các trạm hay các báo cáo hữu ích khác cho người quản trị mạng.

1.1. Phát hiện sự có mặt của Virus

Thông thường, các ứng dụng chạy trên môi trường mạng đều sử dụng các cổng định sẵn (80 cho web, 21 cho FTP, 23 cho Telnet, 110 cho POP3…)



Hình 1. *Mô hình xử lý luồng gói tin*

Như vậy, nếu khi kiểm tra các gói tin có địa chỉ cổng đích khác với các cổng định sẵn ở trên thì rất có khả năng đó là các gói tin chứa Virus. Ví dụ: Virus WinCrash dùng cổng 3024,Virus MicroSpy dùng cổng 3031, Virus Delta Remote Access dùng cổng 3119.

1.2. Phát hiện tấn công từ chối dịch vụ - DoS

Bản chất của tấn công từ chối dịch vụ DoS đó chính là việc rất nhiều máy tính cùng gửi yêu cầu dịch vụ tới một máy nhất định. Nói cách khác là các luồng gói tin sẽ có cổng nguồn giống nhau nhưng đều có cổng đích giống nhau. Như vậy, để phát hiện xem có kiểu tấn công này hay không thì có thể kiểm tra xem tổng số trạm (địa chỉ nguồn) truy cập đến cùng một cổng của địa chỉ đích có vượt qua một ngưỡng cho phép nào đó hay không?. Nếu con số này lớn quá một ngưỡng cho phép thì rất có thể kết luận là bị tấn công DoS.

2. Phân tích luồng gói tin với HiveQL

Với HiveQL, việc xử lý các yêu cầu ở trên khá đơn giản bằng các câu lệnh HiveQL.

Giả thiết thông tin về các luồng được lưu trữ trong một bảng trong Hive là **tblFlows**(SrcIP, DstIP, SrcPort, DstPort) và danh sách Virus đã biết được lưu trữ trong bảng **tblVirus**(VirusPort, VirusName).

- Khi đó, để phát hiện sự có mặt của Virus trong các luồng, có thể dùng câu lệnh (i):

SELECT DISTINCT SrcIP,DstIP, SrcPort, DstPort, Virusname
FROM tblFlows F **JOIN** tblVirus V
ON F.SrcPort=V.VirusPort

- Để phát hiện có tấn công DoS hay không, có thể dùng câu lệnh sau đây để đưa ra các trạm đã sử dụng quá nhiều cổng (Thread) để truy xuất vào cùng một trạm đích. Từ kết quả kết xuất theo thứ tự giảm dần có thể biết được các máy nguồn khả nghi tấn công DoS (ii):

SELECT SrcIP, DstIP, DstPort, Count(*) C
FROM tblFlows **GROUP BY** SrcIP, DstIP, DstPort
ORDER BY C DESC LIMIT 100;

IV. ĐỀ XUẤT CẢI TIẾN HIỆU NĂNG BẰNG HÀM TỰ ĐỊNH NGHĨA

1. Hàm tự định nghĩa trong HiveQL

Ở đây, kích thước của bảng tblFlows thường là rất lớn (big huge), có hàng triệu thậm chí hàng tỉ bản ghi, vì vậy khi xuất câu lệnh kết nối với một bảng khác thì tốc độ xử lý sẽ rất chậm và tiêu tốn

rất nhiều tài nguyên khiến cho hiệu năng chung của toàn hệ thống giảm đáng kể.

Thật may mắn là HiveQL cung cấp một cơ chế cho phép người dùng có thể tự định nghĩa một số hàm và cho phép gọi các hàm này ngay trong câu lệnh HiveQL, kiểu như:

SELECT * FROM tblFlow **WHERE** MyFunction(DstPort) = 80;

Trong đó: MyFunction là hàm do người dùng tự định nghĩa, viết bằng ngôn ngữ Pert, Java…

Cách định nghĩa và gọi hàm tự định nghĩa này có thể tham khảo trong [11].

2. Cải tiến hiệu năng với hàm do người dùng tự định nghĩa

Khi thực hiện kết nối giữa hai bảng ở trên, giả sử kích thước của bảng lớn chứa *N* bản ghi và của bảng nhỏ chứa *k* bản ghi, thì về bản chất, Hive sẽ quét toàn bộ *N* bản ghi và với mỗi bản ghi nhận về sẽ so sánh với *k* bản ghi trong bản nhỏ. Như vậy số phép so sánh mà Hive thực hiện trong câu lệnh SELECT sẽ là *N* x *k* phép. Do đó, với số *N* cực lớn thì dù *k* có nhỏ (cỡ hàng trăm hoặc nghìn bản ghi) thì số phép so sánh cũng sẽ vô cùng lớn.

Để giảm số phép so sánh này, một phương án mà bài báo đề xuất đó là giảm thời gian so sánh giữa một bản ghi nhận được trong *N* bản ghi với *k* bản ghi trong bảng nhỏ thông qua cơ chế hàm tự định nghĩa, trong đó việc so sánh này sẽ mất thời gian là hằng số *O(1)*. Lý do thực hiện được điều này đó là do *k* khá nhỏ nên ta hoàn toàn có thể đưa vào một cấu trúc bảng băm (Hash Table) và như vậy, thời gian tìm kiếm và so sánh chỉ là *O(1)*.

Các bước thực hiện như sau:

Bước 1: Xây dựng một lớp (Class) trong Java.

Bước 2: Nạp bảng nhỏ chứa *k* bản ghi vào một cấu trúc Hash trong hàm khởi tạo

Bước 3: Cài đặt (Overload) hàm **evaluate** trong Java, nhận tham số là giá trị khóa cần kiểm tra, và trả về true nếu thấy khóa đó tồn tại trong Hash và false nếu ngược lại.

Bước 4: Khai báo hàm này với một tên bí danh (ví dụ ***IfExistInTable***) trong Hive và sử dụng.

Như vậy, câu lệnh kiểm tra sự xuất hiện của Virus ở trên:

SELECT DISTINCT SrcIP,DstIP, SrcPort, DstPort, Virusname

FROM tblFlows F **JOIN** tblVirus V **ON** F.SrcPort=V.VirusPort

Có thể viết lại như sau mà không cần kết nối bảng như sau:

SELECT DISTINCT SrcIP,DstIP, SrcPort, DstPort, Virusname

FROM tblFlows **WHERE IfExistInTable**(SrcPort)

Theo cách này, ta có thể áp dụng cho trường hợp kết nối 3 bảng, 4 bảng v.v…

V. THỬ NGHIỆM, ĐÁNH GIÁ

1. Môi trường thử nghiệm

Trong nghiên cứu này, chúng tôi tiến hành 2 thử nghiệm:

+ Thử nghiệm thứ nhất để cho thấy khả năng xử lý dữ liệu rất lớn của hệ thống xử lý trên nền Hadoop so với các công cụ thông thường, ví dụ: Flow Tools.

+ Thử nghiệm thứ hai cho thấy hiệu năng được cải tiến rõ rệt giữa phương án đề xuất của chúng tôi so với tốc độ xử lý thông thường khi có kết nối bảng trong câu lệnh HiveQL.

Môi trường phần cứng và phần mềm dùng thử nghiệm:

• Hadoop cluster: 1,2,4,8 nodes; Data node: Intel Xeon@2.4GHz, 12 GB RAM

• Flow-tools: Version 1.6.8;

2. Kết quả thử nghiệm

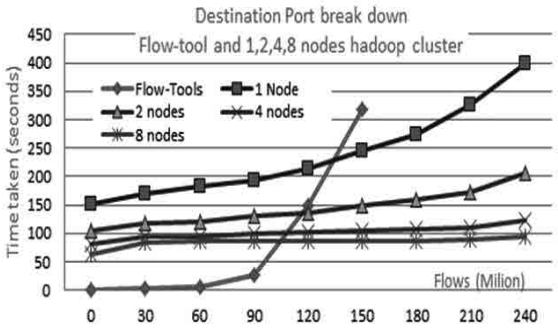
2.1. Kết quả thử nghiệm giữa Flow Tools và Hadoop

Đưa ra danh sách các cổng đích (Destination Port) sử dụng nhiều băng thông nhất (nhiều packets nhất) – Yêu cầu này còn gọi là "Destination Port Break Down":

- Câu lệnh trong công cụ Flow tools là:

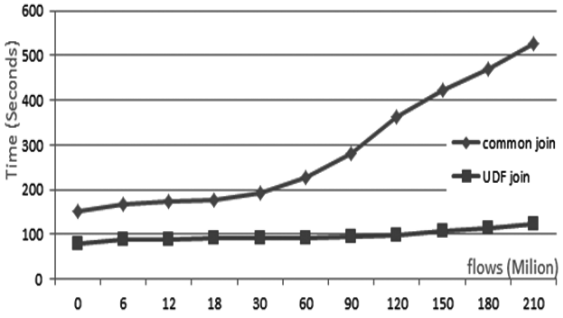
flow-stat –f5 –S2 < LogFile.flow

- Câu lệnh trong HiveQL là: **SELECT dstPort, count(*), sum(packets) P FROM FlowLog GROUP BY dstPort ORDER BY P desc;**

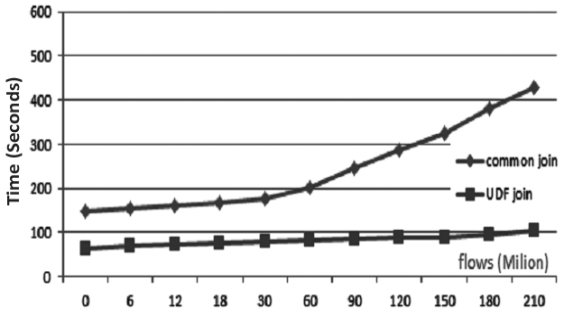


Hình 2. *So sánh khả năng xử lý của công cụ hiện hành với Hadoop*

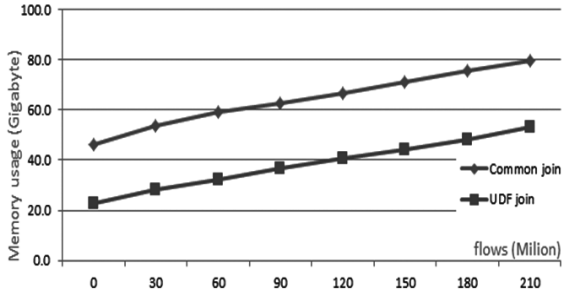
2.2. Kết quả thử nghiệm giữa dùng kết nối thông thường (gọi là Common Join) và kết nối thông qua hàm tự định nghĩa (gọi là UDF Join) sử dụng cụm Cluster 4 nút và 8 nút xử lý



Hình 3. *So sánh tốc độ xử lý khi kết nối bảng trên Cluster 4 nút*



Hình 4. *So sánh tốc độ xử lý khi kết nối bảng trên Cluster 8 nút*



Hình 5. *Tài nguyên bộ nhớ được sử dụng giữa UDF Join và Common Join*

3. Đánh giá

Ở Hình 2 cho thấy, khi dữ liệu nhỏ thì các công cụ xử lý trên máy tính đơn lẻ hiện nay như Flow Tools cho hiệu quả rất tốt và đưa ra kết quả gần như tức thì. Tuy nhiên, khi dữ liệu trở lên rất lớn (trên 150 triệu bản ghi) thì các công cụ này không có khả năng xử lý và bị treo.

Hình 3 và Hình 4, cho thấy hiệu quả xử lý tăng lên rõ rệt khi ta thực hiện kết nối bảng theo đề xuất của bài báo so với kết nối thông thường. Thời gian tăng lên rất chậm ngay cả khi kích thước đầu vào tăng mạnh. Trong khi đó, đối với kết nối thông thường thì thời gian sẽ tăng đột biến khi đầu vào đạt ngưỡng nhất định.

Hình 5 cũng cho thấy tính hiệu quả về mặt tài nguyên bộ nhớ khi sử dụng kết nối thông thường so với kết nối dùng hàm tự định nghĩa theo đề xuất của bài báo.

VI. KẾT LUẬN

Bài báo đã cho thấy rõ khả năng áp dụng nền tảng Hadoop trong việc xử lý dữ liệu lớn là hướng đi rất hứa hẹn và có nhiều ưu thế so với các công cụ sẵn có hiện nay. Đặc biệt, khi kết hợp với Hive thì khả năng khai thác sức mạnh của Hadoop sẽ càng tăng lên và giải quyết được nhiều bài toán thực tế, trong đó việc phân tích luồng gói tin trong mạng phục vụ công tác bảo mật, quản trị mạng là một ứng dụng tiêu biểu.

Đề xuất của bài báo về việc loại bỏ kết nối thông qua cơ chế hàm tự định nghĩa có thể áp dụng để xử lý các trường hợp có kết nối 2 bảng, 3 bảng v.v… ngoài ra cũng có thể áp dụng để xây dựng các câu lệnh quen thuộc như IN, NOT IN trong SQL nhưng HiveQL hiện chưa hỗ trợ.

Tài liệu tham khảo

[1]. http://www.tcpdump.org
[2]. http://www.splintered.net
[3]. http://www.packet-sniffer.net/
[4]. http://www.fukuda-lab.org/mawilab
[5]. http://sortbenchmark.org/
[6]. www.wireshark.org
[7]. http://www.ti.com/tool/packet-sniffer
[8]. https://supportforums.cisco.com/docs/DOC-25009
[9]. http://hadoop.apache.org

[10]. http://hive.apache.org
[11]. http://www.cisco.com/web/go/netflow
[12]. L. Deri, nProbe, "an Open Source NetFlow Probe for Gigabit Networks". *TERENA Networking Conference*, May 2003.
[13]. J. Dean, MapReduce, "*Simplified Data Processing on Large Cluster*", OSDI, 2004.
[14]. Youngseok Lee, Wonchul Kang, "An Internet Traffic Analysis Method with MapReduce". *IEEE/IFIP Network Operations and Management Symposium Workshops*, 2010.
[15]. J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *In Proc. of the 6th Symposium on Operating Systems Design and Implementation*, San Francisco CA, Dec. 2004.
[16]. A. Gates, O. Natkovich, S. Chopra, P. Kamath, S. Narayanam, C. Olston, B. Reed, S. Srinivasan, U. Srivastava. "Building a High-Level Dataflow System on top of MapReduce: The Pig Experience," I*n Proc. of Very Large Data Bases*, **vol 2 no. 2**, 2009, pp. 1414–1425.
[17]. S. Ghemawat, H. Gobioff, S. Leung. "The Google File System," *In Proc. of ACM Symposium on Operating Systems Principles*, Lake George, NY, Oct 2003, pp 29–43.
[18]. T. White, Hadoop, "*The Definitive Guide*". O'Reilly Media, Yahoo! Press, June 5, 2009.

PERFORMANCE ANALYSIS OF PARALLEL COMPUTING ARCHITECTURE USING HADOOP FRAMEWORK

Abstract:

Parallel computing architecture based on PC cluster systems using Hadoop Framework in the past few years received great attention because of the computing power and expansion of the system is quite favorable and easy, especially is in handling extremely large data amounting to Tera-bytes or even Peta-bytes. However, the performance of this architecture depends on many factors, such as installing algorithm, HiveQL queries optimization, cluster architectures, etc ... This paper proposes solutions using parallel computing architecture Hive on Hadoop Framework for processing network packet flows in order to detect the presence of viruses and analyzed and givea number of solutions to improve performance of the computing system.

Keywords: *Hadoop, HiveQL, Parallel processing, packet flow analysis, performance analysis.*