Khoa Công nghệ Thông tin

Hệ Thống Phân Tán - 18CNTN

BÁO CÁO ĐỒ ÁN
Schiper-Eggli-Sandoz Algorithm

18120143 Nguyễn Đình Hoàng Phúc

Ngày 17 tháng 10 năm 2021

Tóm tắt nội dung

Đồng bộ đồng hồ là một yếu tố quan trọng để hệ thống phân tán có thể hoạt động đúng, có thứ tự và tổ chức. Có hai loại đồng hộ là đồng hồ vật lý và đồng hộ logic. Đồng bộ đồng hộ vật lý là điều vô cùng khó khăn và gần như không thể đạt được độ chính xác tuyệt đối. Do đó, đồng hồ logic được tạo ra như một bộ đánh số các sự kiện xảy ra, đảm bảo tính thứ tự hay quan hệ giữa các sự kiện. Có rất nhiều cấu trúc dữ liệu và giao thức để mô tả và cập nhật đồng hộ giữa các nút trong hệ thống, trong đó Schiper-Eggli-Sandoz (SES) là một giao thức đơn giản nhưng hiệu quả để mô tả đồng hồ, đảm bảo quan hệ nhân quả của các sự kiện xảy ra trên các máy tính trong hệ thống. Trong đồ án này, em đã tìm hiểu về thuật toán, thực thi thuật toán bằng ngôn ngữ lập trình Python và chạy trên localhost để minh họa.

1. Tóm tắt thuật toán

1.1. Dữ liệu lưu trữ:

Gọi n là số máy trong hệ thống. Tại máy thứ i lưu các thông tin sau:

- $\mathbf{t}_{P_i}$: thời điểm logic.
- $\mathbf{V}_{P_i}$: vector có kích thước $n-1$, mỗi phần tử là tuple có dạng $(P_j, \mathbf{t})$ với P_j $(i \neq j)$ là **ID** của máy khác máy đang xét, và $\mathbf{t}$ là một vector clock. Xét một thời điểm nào đó, $\mathbf{V}_{P_i}$ cho biết thời gian logic của các

máy khác mà máy i đã biết được và cập nhật cho đến thời điểm này. Cụ thể hơn là cho biết tại thời điểm đang xét đã có bao nhiêu sự kiện diễn ra ở các máy khác mà máy i đã biết.

1.2. **Dữ liệu gửi thêm trong gói tin:**

Gọi một gói tin nào đó là m, giả sử m được gửi từ máy i, m lưu các thông tin sau:

- $\mathbf{t}_m$: thời điểm logic tại máy gửi lúc m gửi đi.
- $\mathbf{V}_m$: vector $\mathbf{V}_{P_i}$ tại thởi điểm m được gửi đi.

1.3. **Gửi:**

Giả sử máy i cần gửi gói tin tới máy j, thực hiên các thao tác sau:

- Bước 1: Cập nhật lại $\mathbf{t}_{P_i}$
- Bước 2: Tạo dữ liệu cho gói tin được gửi.
- Bước 3: Thêm hoặc ghi đè lại $(P_j, \mathbf{t})$ trong $\mathbf{V}_{P_i}$ thành $(P_j, \mathbf{t}_{P_i})$. Bước này nhằm lưu lại sau khi gửi gói tin này, máy i biết được rằng sau khi nhận, đồng hồ tại máy j sẽ phải lớn hơn hoặc bằng $\mathbf{t}_{P_i}$)

1.4. **Nhận:**

Giả sử máy i đã gửi gói tin tới máy j, khi máy j nhận thực hiên các thao tác sau:

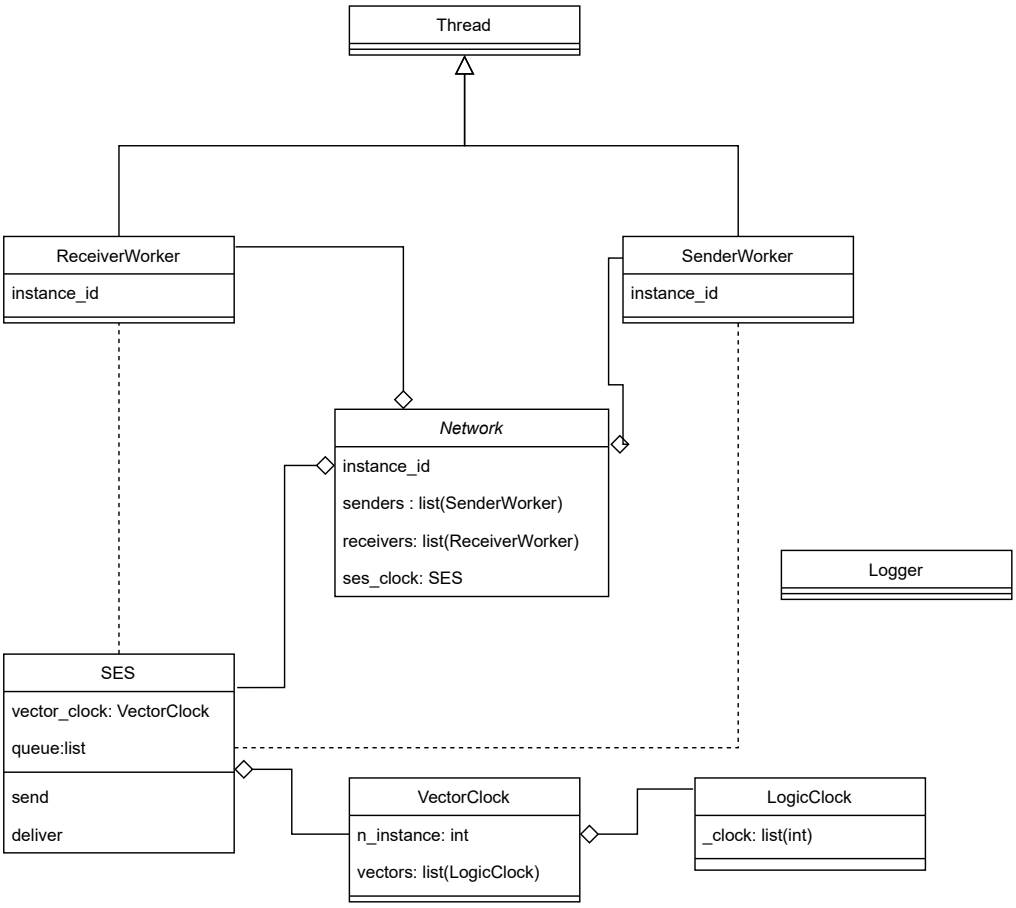
- Bước 1: Kiểm tra, nếu trong $\mathbf{V}_m$ của gói tin không chứa $(P_j, \mathbf{t})$ thì chuyển tới bước 3.
- Bước 2: So sánh $(P_j, \mathbf{t})$ trong $\mathbf{V}_m$ với $\mathbf{t}_{P_j}$, nếu $\mathbf{t} < \mathbf{t}_{P_j}$, cho phép chuyển gói tin, ngược lại lưu vào buffer và chuyển tới bước 4. Điều kiện này không thỏa cho biết rằng máy j chưa kịp cập nhật đồng hồ đến thời điểm mà trước khi gửi gói tin máy i "mong đợi"(đồng nghĩa với việc chưa nhận đủ gói tin). Do đó khi điều kiện này không thỏa ta cần lưu lại gói tin này để xử lý sau.
- Bước 3: Cập nhật. Gồm cập nhật lại $\mathbf{t}_{P_j}$ dựa trên $\mathbf{t}_m$ theo thuật toán cập nhật vector clock thông thường và cập nhật $\mathbf{V}_{P_i}$ dựa theo $\mathbf{V}_m$ bằng cách: xét từng phần tử trong $\mathbf{V}_m$, gọi là $(P_k, \mathbf{t})$
  - Nếu $(P_k, \mathbf{t})$ không có trong $\mathbf{V}_{P_i}$ và $k \neq j$ thì thêm $(P_k, \mathbf{t})$ vào $\mathbf{V}_{P_i}$
  - Nếu $(P_k, \mathbf{t})$ có trong $\mathbf{V}_{P_i}$ và $k \neq j$ thì cập nhật lại bằng cách lấy giá trị lớn nhất từng phần tử (maximum-wise).

- Bước 4: Xét lại các phần tử trong buffer, thực hiện lại các thao tác kiểm tra và cập nhật ở bước 2 và 3.

2. Cấu trúc chương trình

Chương trình được viết sử dụng ngôn ngữ python version Python 3.6.15 gồm các module sau:

- ./Network: chứa mã nguồn hỗ trợ cho việc kết nối mạng các tiến trình.
- ./SES: chứa mã nguồn của thuật toán SES.
- ./Log: chứa mã nguồn hỗ trợ việc in log của chương trình.
- ./static: output các file log của chương trình.
- ./main.py: chương trình chính để khởi động một process.
- ./constant.py: chứa các hằng số.
- ./run.sh: bash script để chạy 15 process.



Hình 1: Sơ đồ lớp của chương trình.

3. Chi tiết cài đặt

3.1. Kết nối mạng:

Quy ước rằng, nếu có n process các process sẽ được đánh số từ 0 đến $n-1$ gọi là instance-id. Mỗi process sẽ chiếm port bằng với 60000+instanced-id. Có thể thay đổ số 60000 này trong file constant.py.

Khi khởi động chương trình. Process được tạo sẽ listen trên một port định trước (dựa vào id của máy trong hệ thống). Khi có bất cứ yêu cầu kết nối nào, process tạo một thread mới để đảm nhiệm việc trao đổi trên connection này. Song Song đó, process cũng khởi tạo $n-1$ threads để làm senders, gửi request mở kết nối đến $n-1$ processes còn lại trong hệ thống. Sau khi một sender kết nối thành công sẽ tiến hành gửi các gói tin, ngược lại nếu chưa kết nối thành công (do process khác chưa khởi động) thì sẽ gửi request mở kết nối lại sau 0.5 giây. Cứ như vậy đến một thời điểm hệ thống sẽ kết nối hoàn toàn với nhau và gửi nhận các gói tin có chứa thông tin về đồng hồ để thực hiện việc đồng bộ.

Lưu ý, để dễ xảy ra trường hợp buffer, receiver không xử lí gói tin ngay, mà gây nhiễu bằng cách đảo thứ tự theo xác suất.

3.2. Thuật toán SES:

Về cách lưu trữ:

- class LogicClock: là một vector các số nguyên có n phần tử.
- class VectorClock: dữ liệu lưu trữ của thuật toán SES (xem lại mục 1.1), cụ thể V_P và t_P đều được lưu trong cùng một vector n phần tử, mỗi phần tử là một LogicClock. Có thể dễ dàng truy suất từng timestamp của từng process thông qua instance-id.
- class SES: một đối tượng SES chứa dữ liệu lưu trữ của thuật toán SES và buffer.

Về cách thực thi, do thông tin cần được gửi dưới dạng nhị phân, các lớp trong module SES đều chứa các phương thức để serialize và deserialize, tức biểu diễn lớp dưới dạng nhị phân và khởi tạo lại đối tượng từ nhị phân. Bên cạnh đó các phương thức trong các module có rất nhiều phương thức hỗ trợ, ở đây chỉ trình bày về hai phương thức quan trọng là deliver và send. Do cả hai thao tác này đều thực hiện việc chỉnh sửa thông tin chung, và

được thực hiện trên nhiều thread, nên để đảm bảo đồng bộ, chương trình sử dụng **semaphore** để đảm bảo tính atomic của từng thao tác.

- Phương thức send:
  - *self.vector_clock.increase()*: Tăng clock $t_P[i]$ lên một đơn vị.
  - *result = self.serialize(packet)*: Thêm vào packet gửi thông tin đồng hồ.
  - *self.vector_clock.self_merge(self.vector_clock.instance_id, destination_id)*: Gán t_P vào V_P tại vị trí của process nhận tin.
- Phương thức deliver:
  - *source_vector_clock, packet = self.deserialize(packet)*: Tách đồng hồ trên gói tin thành đối tượng để xử lí.
  - *if t_m < t_p*: kiểm tra điều kiện để deliver gói.
  - *self.merge(source_vector_clock)*, thực hiện bước 3 của thuật toán ở phần 1.4, bao gồm cập nhật lại các phần tử trong V_P và cập nhật lại t_P.

3.3. Log:

Chương trình sử dụng ba loại log, và in log ra hai đích là màn hình và file. Ba loại log bao gồm, log gửi gói tin (in gói tin gửi) tương ứng file ##__sender.log, log nhận gói tin (in gói tin nhận) tương ứng file ##__receiver.log và log chung (in các thông tin chung) tương ứng file ##__general.log. Chỉ có log nhận tin và log chung được in ra màn hình.

4. Hướng dẫn sử dụng

Chương trình được viết chạy và kiểm tra hoạt động trên hệ điều hành Linux, distribution Ubuntu 20.04.3 LTS, python Python 3.6.15.

- Bước 1: Mở thư mục chứa chương trình.
- Bước 2: Chạy bằng lệnh "sudo bash ./run.sh <số-process-cần-chạy>"
- Bước 3: Quan sát log nhận tin trên màn hình, hoặc quan sát rõ các log trong thư mục ./static/log.
- Bước 4: Tắt chương trình bằng "Ctrl+C", cần tắt thủ công từng process, chương trình đã xử lí được việc một process ngừng hoạt động và thực hiện việc đóng các kết nối, giải phóng port,...

5. Kết quả

Kết quả chạy thử được gửi kèm trong ./static/ gồm các kết quả chạy thử chương trình trên 2, 3 và 15 process.

6. Tài liệu tham khảo

1. Slide bài giảng môn Hệ thống phân tán khóa 2018.
2. Tài liệu về Python socket.
3. Bài báo: A new algorithm to implement causal ordering.