TẠP CHÍ KHOA HỌC
TRƯỜNG ĐẠI HỌC SƯ PHẠM TP HỒ CHÍ MINH
Tập 20, Số 7 (2023): 1166-1179
Website: https://journal.hcmue.edu.vn
ISSN: 2734-9918

HO CHI MINH CITY UNIVERSITY OF EDUCATION
JOURNAL OF SCIENCE
Vol. 20, No. 7 (2023): 1166-1179
https://doi.org/10.54607/hcmue.js.20.7.3653(2023)

Bài báo nghiên cứu

PHÂN TÍCH VÀ KIỂM CHỨNG KIẾN TRÚC HAYSTACK TRONG MẠNG XÃ HỘI FACEBOOK

Lê Thị Thúy, Bùi Quốc Việt*

Trường Đại học Sư phạm Thể dục Thể thao Thành phố Hồ Chí Minh, Việt Nam

**Tác giả liên hệ: Bùi Quốc Việt – Email: vietqb@upes.edu.vn*

Ngày nhận bài: 11-10-2022; ngày nhận bài sửa: 05-11-2022; ngày duyệt đăng: 26-6-2023

TÓM TẮT

Haystack là một kiến trúc hệ thống lưu trữ được tối ưu hóa cho ứng dụng ảnh của Facebook. Haystack có bốn ưu điểm chính so với hệ thống trước đó bao gồm, thông lượng cao và độ trễ thấp, khả năng chịu lỗi, chi phí hiệu quả và tính đơn giản. Với việc sử dụng rộng rãi của kiến trúc Haystack trong Facebook, thì tính hợp lệ của nó và các thuộc tính chính yếu khác được trừu tượng hóa từ kiến trúc này cần phải được phân tích và kiểm chứng trong một tiếp cận chính xác. Bài viết tập trung vào thiết kế bên trong việc xử lí và tải nạp một bức ảnh của kiến trúc Haystack và áp dụng đại số tiến trình CSP để phân tích chúng một cách chi tiết. Bằng cách đưa các mô hình vào bộ công cụ phân tích tiến trình PAT để kiểm chứng một số tính chất quan trọng, bao gồm tính chất cơ bản và tính chất bổ sung. Tính chất cơ bản bao gồm Deadlock Freedom; các tính chất bổ sung bao gồm truy cập tương tranh, truy cập tương tranh không đồng bộ, truy cập tương tranh với cùng một máy khách, tải nạp tương tranh và tải nạp tương tranh với cùng một máy khách. Cuối cùng, theo kết quả kiểm chứng, chúng tôi thấy rằng từ góc độ CSP, các tính chất của kiến trúc Haystack là hợp lệ, có nghĩa là nó đáp ứng các yêu cầu theo tài liệu của Facebook.

Từ khóa: phân tích; CSP; Haystack, kiểm chứng; PAT

1. Giới thiệu

Sự phổ biến của các mạng xã hội đã thúc đẩy sự gia tăng số lượng người dùng được tạo ra bởi người dùng Internet. Chia sẻ hình ảnh là một trong những tính năng phổ biến nhất trên mạng xã hội Facebook. Với mỗi ảnh tải lên, Facebook tạo ra và lưu trữ 4 kích thước khác nhau của chúng. Số lượng ảnh người dùng tải lên sẽ tiếp tục tăng lên trong tương lai và tạo ra một thách thức lớn đối với cơ sở hạ tầng của Facebook. Do đó, tính hiệu quả của ngăn xếp lưu trữ và truyền các đối tượng nhị phân lớn (Blob - Binary large object) đã trở thành một vấn đề quan trọng đối với cộng đồng các nhà cung cấp mạng xã hội (Doug Beaver, 2010). Khi nội dung số được sử dụng ngày nhiều hơn, các nền tảng mạng xã hội nâng cấp kiến trúc tệp của họ để đối phó với những tin tức cập nhật mới nhất, chẳng hạn như mạng xã

Cite this article as: Le Thi Thuy, & Bui Quoc Viet (2023). Analysis and formalization of Haystack architecture in Facebook. *Ho Chi Minh City University of Education Journal of Science, 20*(7), 1166-1179.

hội Facebook. Do đó, sự ổn định của hệ thống lưu trữ tệp tin trên nền tảng mạng xã hội là rất cần thiết.

Dựa trên các tài liệu chính thức liên quan đến Haystack, các tính chất của Haystack sẽ được trừu tượng hóa, và sau đó là các thuộc tính chính của Haystack được phân tích và kiểm chứng bằng cách sử dụng CSP và PAT trong bài viết này.

2. Đối tượng và phương pháp nghiên cứu

Chúng tôi áp dụng đại số tiến trình CSP để phân tích kiến trúc Haystack; cài đặt và kiểm chứng các tính chất của kiến trúc Haystack bằng bộ công cụ phân tích tiến trình PAT.

2.1. Ngôn ngữ đại số CSP

Đại số tiến trình sử dụng các phương pháp đại số để nghiên cứu việc truyền thông của các hệ thống tương tranh. Có ba hệ thống lập luận điển hình gồm Communicating Sequential Processes (CSP), Algebra of Communicating Processes (ACP) (Jan A. Bergstra, & Jan Willem Klop, 1985) và Calculus of Communicating Systems (CCS) (Robin Milner, 1980). CSP, là một đại số tiến trình (Brookes, Hoare, & Roscoe, 1984; Hoare, 1978), do Hoare đề xuất vào năm 1978. Ngôn ngữ đại số này được thiết kế chủ yếu để mô tả và phân tích hành vi của các hệ thống và tiến trình tương tranh (Lowe & Roscoe, 1997; Roscoe, 2010, Ngo, 2011).

Các tiến trình CSP được cấu thành bởi các tiến trình và hành động ban đầu. Cú pháp của tập con ngôn ngữ CSP để xác định các tiến trình, theo đó P và Q đại diện cho các tiến trình, bảng chữ cái $\alpha(P)$ và $\alpha(Q)$ có nghĩa là tập hợp các hành động mà các tiến trình P và Q có thể thực hiện tương ứng, a và b biểu thị các hành động ban đầu và c là viết tắt cho tên của một kênh. Sau đó, cú pháp cơ bản của CSP được định nghĩa như sau:

$$P, Q = Skip \mid Stop \mid a \to P \mid c?x \to P \mid c!e \to P \mid$$
$$P \,\square\, Q \mid P \,||\, Q \mid P \,|||\, Q \mid P \lhd b \rhd Q \mid P \,;\, Q \mid P \,[|X|]\, Q$$

ở đây:

- *Skip* chỉ tiến trình không làm gì nhưng kết thúc thành công; *Stop* chỉ tiến trình trong trạng thái bế tắc và không làm gì;
- $P \,\square\, Q$ chỉ sự lựa chọn tổng quát giữa tiến trình P hoặc tiến trình Q; $P \,||\, Q$ chỉ một tiến trình gồm hai tiến trình song song giữa tiến trình P và tiến trình Q; $P \,|||\, Q$ mô tả hai tiến trình chạy đồng thời mà không có đồng bộ hóa, trong đó $|||$ biểu thị sự xen kẽ; $P \lhd b \rhd Q$ chỉ sự lựa chọn có điều kiện. Nếu giá trị của b là đúng thì nó hoạt động như P ngược lại hoạt động như Q.

2.2. Bộ công cụ phân tích tiến trình PAT

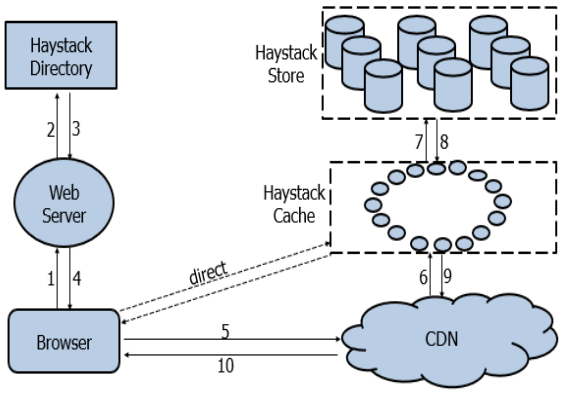
PAT, là viết tắt của Process Analysis Toolkit (National University of Singapore, 2008), một công cụ dựa trên CSP và được thiết kế để áp dụng các kĩ thuật kiểm chứng mô hình để phân tích hệ thống tự động. Dưới đây, là một số kí hiệu trong PAT:

- *#define N* 0 định nghĩa 1 hằng số toàn cục *N* với giá trị ban đầu là 0.

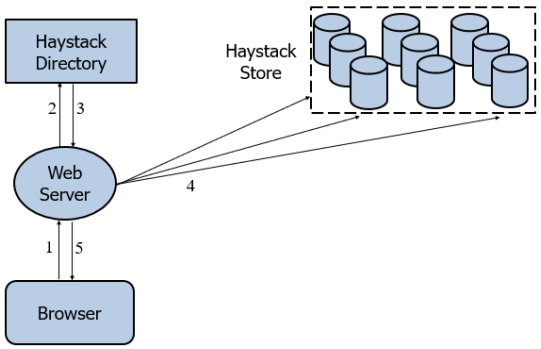
- *channel c* 1 phát biểu này khai báo một kênh, *c* là tên kênh và 1 là kích thước bộ đệm. *var cond = false* biểu thị điều kiện boolean với giá trị ban đầu là *false*.
- [*cond*] *P* là viết tắt của một tiến trình được bảo vệ.
- *#define goal n* > 0; *#assert P reaches goal*; điều này định nghĩa một xác nhận và nó kiểm tra xem tiến trình *P* có thể đạt tới trạng thái mà điều kiện *goal* được thỏa mãn hay không.
- *||| i: 0..N@ P(i)* xác định *N* tiến trình, bao gồm *P(1), P(2), ..., P(N)*, được chạy bằng cách xen kẽ với nhau.
- *#assert P () | = F*; định nghĩa một xác nhận kiểm tra xem tiến trình *P* có thỏa mãn công thức *F*.

2.3. Giới thiệu về kiến trúc Haystack

Kiến trúc Haystack bao gồm tải xuống một ảnh và tải lên một ảnh. Haystack là một kiến trúc hệ thống lưu trữ mà Facebook thiết kế để chia sẻ ảnh, có thể hoạt động tốt hơn bất kỳ hệ thống lưu trữ truyền thống nào dưới khối lượng công việc tương đương. Kiến trúc của Haystack gồm ba thành phần cốt lõi: Haystack Store, Haystack Directory và Haystack Cache.



Hình 1. *Tải xuống một ảnh*



Hình 2. *Tải lên một ảnh*

2.3.1. Quy trình làm việc của kiến trúc Haystack

Vai trò của Haystack là lưu trữ và chia sẻ các tệp nhỏ như ảnh, do đó kiến trúc tổng thể có thể được chia nhỏ hơn nữa để tải xuống một ảnh và tải lên một ảnh.

Tải xuống một ảnh. Hình 1 miêu tả các thành phần Store, Directory và Cache kết hợp với nhau để phục vụ truy vấn từ Browser của người dùng, Web Server, CDN và hệ thống lưu trữ.

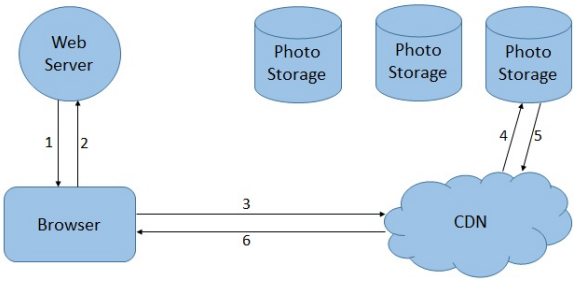
Gửi đến Cache. Nếu hình ảnh là phổ biến nhất, Browser sẽ gửi thông tin URL trực tiếp đến Cache.

Gửi đến CDN. Nếu hình ảnh ít phổ biến hơn, Browser sẽ gửi thông tin URL trực tiếp đến CDN.

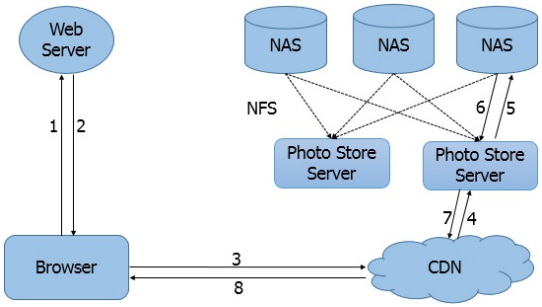
Tải lên một ảnh. Khi tải ảnh lên, chỉ có bốn thành phần trong kiến trúc này, đó là Browser, Web Server, Directory và Store được hiển thị trong Hình 2.

2.3.2. So sánh kiến trúc Haystack với thiết kế điển hình và thiết kế dựa trên giao thức của hệ thống tập tin phân tán NFS (Network File System)

Hình 3 mô tả các bước từ khi người dùng truy cập một trang web có chứa hình ảnh cho đến khi họ tải nó xuống ổ đĩa máy tính.



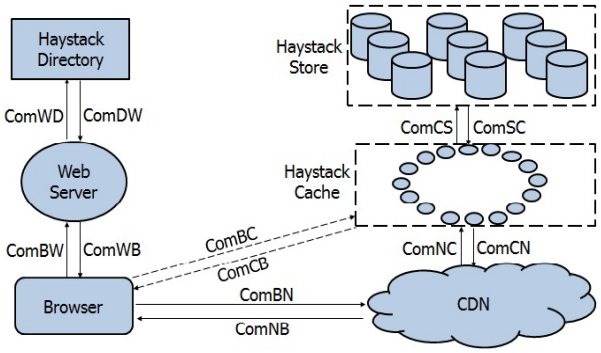
Hình 3. *Thiết kế điển hình*



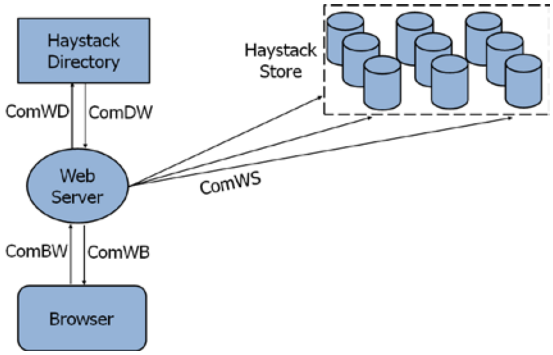
Hình 4. *Thiết kế dựa trên NFS*

Hình 4 mô tả các bước của thiết kế dựa trên NFS. Thiết kế này lưu trữ từng ảnh trong tệp riêng của nó gắn trên một bộ thiết bị NAS. Sau đó, một tập hợp máy chủ lưu trữ ảnh (Photo Store Server) gắn kết tất cả các ổ đĩa được xuất bởi các thiết bị NAS này qua NFS.

So với kiến trúc trước đó, Haystack khung được cải tiến, cho thấy sự khác biệt giữa đồng thời thông lượng cao, có nghĩa là thiết kế điển hình và thiết kế dựa trên NFS đọc và ghi nhiều lần hơn so với khung Haystack.



Hình 5. *Các kênh tải xuống một ảnh*



Hình 6. *Các kênh tải lên một ảnh*

3. Kết quả và thảo luận

3.1. Phân tích kiến trúc Haystack

Trong phần này, chúng tôi sẽ mã hóa kiến trúc Haystack trong CSP. Kiến trúc Haystack có ba thành phần cốt lõi: Store, Directory và Cache. Sau đây chúng tôi sẽ đi vào phân tích cụ thể các thành phần.

3.1.1. Thông điệp và kênh

Hình 5 và Hình 6 cung cấp các kênh truyền thông. Để thuận tiện hơn, chúng tôi đưa ra định nghĩa các tập hợp được sử dụng trong các mô hình: tập hợp các **Browser** của thành phần Trình duyệt, **WebServer** của thành phần Máy chủ Web, **CDN** của thành phần CDN, **Cache** của Bộ nhớ đệm, **Directory** của Thư mục và **Store** của thành phần Lưu trữ, **REQ** của yêu cầu, **URL** của thông điệp url và **Data** của thông tin ảnh.

Dựa trên các tập hợp được xác định ở trên, chúng tôi xác định các thông điệp được truyền giữa các thành phần như sau:

$$
\begin{aligned}
MSG &= MSG_{req} \cup MSG_{rep} \cup MSG_{data} \\
MSG_{req} &= \{msg_{req}.A.B.content | A \in Browsers \\
&\quad \cup WebServer \cup CDN \cup Cache, \\
&\quad B \cup WebServer \cup CDN \cup Cache \\
&\quad \cup Store \cup Directory, content \in REQ\} \\
MSGrep &= msg_{rep}.A.B.content | A \in WebServer \cup \\
&\quad CDN \cup Cache \cup Store \cup Directory, \\
&\quad B \in Browsers \cup WebServer \cup CDN \\
&\quad \cup Cache, content \in URL \cup REQ\} \\
MSG_{data} &= \{msg_{data}.A.B.content | A \in CDN \cup Cache \cup Store, \\
&\quad B \in Browsers \cup CDN \cup Cache; content \in Data\}
\end{aligned}
$$

trong đó, MSG_{req} đại diện cho tập hợp các thông điệp yêu cầu, MSG_{rep} là viết tắt của tất cả các loại yêu cầu phản hồi và MSG_{data} đại diện cho tập hợp các thông điệp truyền dữ liệu ảnh thực. Mỗi thông điệp chứa một thẻ từ tập hợp $\{msg_{req}, msg_{rep}, msg_{data}\}$. Sau đó, chúng tôi đưa ra định nghĩa của các kênh. Trong bài viết này, các kênh sử dụng **COM_PATH** để biểu diễn và được xác định như sau:

ComWD, ComDW, ComBW, ComWB, ComBN, ComNB, ComNC,
ComCN, ComCS, ComSC, ComBC, ComCB, ComWS, ComSW.

Khai báo của các kênh như sau:

Kênh *COM_PATH*: *MSG*

Bảng 1. *Giải thích các thông điệp của mô hình*

Thông điệp	Chức năng
req_serv	tương tự như yêu cầu HTTP mang ID ảnh
photoID	ID của ổ đĩa vật lí máy được lưu trữ ảnh
physicalID	ID của ảnh được phân phát hoặc tải lên
logicalID	ID của ổ đĩa logic máy được lưu trữ ảnh
cookie	cookie được WebServer phân bổ ngẫu nhiên
state	trạng thái về mức độ phổ biến của bức ảnh
data	dữ liệu của bức ảnh
none	không có gì hoặc thất bại
complete	thành công

3.1.2. Hệ thống (System)

Toàn bộ hệ thống mô hình kiến trúc Haystack bao gồm hai phần, đó là $System_{serv}$ và $System_{upload}$. Cả hai đều xen kẽ. $System_{serv}$ gồm sáu tiến trình con, bao gồm $Browser_{serv}$, $WebServer_{serv}$; $Directory_{serv}$, *CDN*, *Cache* và $Store_{serv}$, tương ứng, đang chạy song song;

$System_{upload}$ bao gồm bốn tiến trình con, lần lượt là $Browser_{upload}$, $Webserver_{upload}$, $Directory_{upload}$ và $Store_{upload}$, đang chạy song song. Hệ thống được phân tích như sau:

$$SYSTEM(b,s) =_{df} System_{serv}(b,s) \;|||\; System_{upload}(b,s)$$
$$System_{serv}(b,s) =_{df} Browser_{serv}(b,s) \;||\; WebServer_{serv}(b) \;||\; CDN(b,s) \;||\; Cache(b,s) \;||\; Directory_{serv} \;||\; Store_{serv}(b,s)$$
$$System_{upload}(b,s) =_{df} Browser_{upload}(b) \;||\; Webserver_{upload}(b) \;||\; Store_{upload}(s) \;||\; Directory_{upload}$$

3.1.3. Trình duyệt (Browser)

Trong kiến trúc Haystack, có một số Browser như máy khách trong hệ thống phân tán, Browser này gửi yêu cầu dịch vụ và tải ảnh lên. Trên thực tế, Browser là khách truy cập, có thể là điện thoại di động, thiết bị đầu cuối iPad, máy tính và các thiết bị đầu cuối không dây khác. Vì vậy, tất cả các thiết bị đầu cuối này đều được trừu tượng hóa thành tiến trình Browser, tiến trình này chủ yếu bao gồm hai tiến trình con, $Browser_{serv}$ và $Browser_{upload}$. Mô hình cụ thể được phân tích như sau:

$$Browser(b,s) =_{df} Browser_{serv}(b,s) \;|||\; Browser_{upload}(b)$$
$$Browser_{serv}(b,s) =_{df} ComBW!msg_{req}.B.b.W.req\_serv$$
$$\rightarrow ComWB?msg_{rep}.W.B.b.state.photoID.physicalID.logicalID$$
$$\rightarrow \textit{if } state = 1 \textit{ then}$$
$$ComBC!msg_{req}.B.b.C.photoID.physicalID.logicalID$$
$$\rightarrow ComCB?msg_{data}.C.B.b.data \rightarrow Browser_{serv}(b,s)$$
$$\textit{else}$$
$$ComBN!msg_{req}.B.b.N.photoID.physicalID.logicalID$$
$$\rightarrow ComNB?msg_{data}.N.B.b.data \rightarrow Browser_{serv}(b,s)$$
$$Browser_{upload}(b) =_{df} ComBW!msg_{data}.B.b.W.data \rightarrow$$
$$\textit{if } write\_enabled = 1 \;\&\&\; flag = 5 \textit{ then}$$
$$ComWB?msg_{rep}.W.B.b.complete \rightarrow Skip$$
$$\textit{else}$$
$$ComWB?msg_{rep}.W.B.b.none \rightarrow Skip$$

3.1.4. Máy chủ Web (WebServer)

WebServer trong kiến trúc Haystack, giống như một trạm gần nhất được sử dụng để chuyển tiếp thông điệp và dữ liệu. Nó kết nối ba thành phần chính của Haystack và các yêu cầu hoạt động của máy khách. Tương tự, WebServer bao gồm hai tiến trình con, $WebServer_{serv}$ và $WebServer_{upload}$.

Mô hình cụ thể được phân tích như sau:

$$WebServer(b,s) =_{df} WebServer_{serv}(b) \;|||\; WebServer_{upload}(b,s)$$
$$WebServer_{serv}(b) =_{df} ComBW?msg_{req}.B.b.W.req\_serv \rightarrow$$
$$ComWD!msg_{req}.W.D.req\_serv \rightarrow$$
$$ComDW?msg_{rep}.D.W.state.photoID.physicalID.logicalID \rightarrow$$

ComWB!msg_{rep}.*W.B.b.state.photoID.physicalID.logicalID*
→ $WebServer_{serv}(b)$
$WebServer_{upload}(b,s)$ =*df ComBW*?msg_{data}.*B.b.W.data* →
ComWD!msg_{req}.*W.D.data.photoID.physicalID.logicalID* →
if write_enabled = 1 && *flag* = 4 *then*
ComDW?msg_{rep}.*D.W.data.photoID.physicalID.logicalID*
→ *setCookie*(); *setFlag3*();
ComWS!msg_{data}.*W.S.s.data.cookie*
→ *ComSW*?msg_{rep}.*S.s.W.complete*
→ *ComWB*!msg_{rep}.*W.B.complete* → *Skip*
else
ComDW?msg_{req}.*D.W.none*
→ *ComWB*!msg_{rep}.*W.B.b.none* → *Skip*

3.1.5. Thư mục (Directory)

Directory được sử dụng để tạo ánh xạ ổ đĩa logic và ổ đĩa vật lí, đồng thời để xác định xem bức ảnh có phải là bức ảnh phổ biến nhất hay không. Directory bao gồm $Directory_{serv}$ và $Directory_{upload}$.

Trạng thái của mô-đun Directory được phân tích như sau:
Directory =*df* $Directory_{serv}$ ||| $Directory_{upload}$
$Directory_{serv}$ =*df ComWD*?msg_{req}.*W.D.req_serv*
→ *DJudge*(); *ComDW*!msg_{rep}.*D.W.state.photoID.physicalID.logicalID*
→ $Directory_{serv}$
$Directory_{upload}$ =*df*
ComWD?msg_{req}.*W.D.data.photoID.physicalID.logicalID*
→ *Compute*(); *Allocate*();
if write_enabled = 1 *then*
setFlag4(); *ComDW*!msg_{rep}.*D.W.data.photoID.physicalID.logicalID*
→ *Skip*
else
ComDW!msg_{rep}.*D.W.none* → *Skip*

3.1.6. Mạng phân phối nội dung (CDN)

CDN tương đương với hệ thống bên ngoài, được sử dụng chủ yếu để lưu ảnh vào bộ nhớ đệm, ít phổ biến hơn. Chức năng chính của CDN là tải xuống một ảnh.

Trạng thái của CDN được phân tích như sau:
CDN(b,s) =*df ComBN*?msg_{req}.*B.b.N.photoID.physicalID.logicalID*
→ *NJudge*();
if data! = *none then*
ComNB!msg_{data}.*N.B.b.data* → *CDN(b,s)*

else

ComNC!msg_{req}.*N.C.photoID.physicalID.logicalID*

→ *ComCN*?msg_{data}.*C.N.data* →

ComNB!msg_{data}.*N.B.b.data* → *CDN(b,s)*

3.1.7. Bộ nhớ đệm (Cache)

Cache tương tự như CDN, nhưng tương đương với hệ thống bên trong, chủ yếu để Cache lưu những hình ảnh phổ biến nhất. Như trên, chức năng của Cache vẫn là thực hiện tải xuống một ảnh.

Trạng thái của Cache được phân tích như sau:

Cache(b,s) =$_{df}$ *if state* = 1 *then*

ComBC?msg_{req}.*B.b.C.photoID.physicalID.logicalID*

→ *CJudge()*;

if data! = *none then*

ComCB!msg_{data}.*C.B.b.data* → *Browser*$_{serv}$*(b,s)*

else

setFlag1();

ComCS!msg_{req}.*C.S.s.photoID.physicalID.logicalID*

→ *ComSC*?msg_{data}.*S.C.data* →

ComCB!msg_{data}.*C.B.b.data* → *Browser*$_{serv}$*(b,s)*

else

ComNC?msg_{req}.*N.C.photoID.physicalID.logicalID*

→ *CJudge()*;

if data ! = *none then*

ComCN!msg_{data}.*C.N.data ! Browserserv(b,s)*

else

ComCS!msg_{req}.*C.S.s.photoID.physicalID.logicalID*

→ *ComSC*?msg_{data}.*S.C.data* →

ComCN!msg_{data}.*C.N.data* → *CDN(b,s)*

3.1.8. Lưu trữ (Store)

Trên thực tế, Store trong toàn bộ kiến trúc Haystack là sự phân phối của các máy chủ trên khắp thế giới. Chức năng chính của nó là lưu trữ và truy xuất hình ảnh. Tiến trình Store bao gồm hai tiến trình con, *Store*$_{serv}$ và *Store*$_{upload}$.

Trạng thái của Store được phân tích như sau:

Store(b,s) =$_{df}$ *Store*$_{serv}$*(b,s)* ||| *Store*$_{upload}$*(s)*

Store$_{serv}$*(b,s)* =$_{df}$

if ((state = 0 && *flag* = 2) || *state* = 1 && *flag* = 1))*then*

ComCS?msg_{req}.*C.S.s.photoID.physicalID.logicalID*

→ *SJudge()*; *ComSC*!msg_{data}.*S.s.C.data* → *Cache(b,s)*

else
Store$_{serv}$*(b,s)*
Store$_{upload}$*(s)* =$_{df}$
if (write_enabled = 1 && *flag* = 3*) then*
ComWS?*msg*$_{data}$*.W.S.s.data.cookie* →
setFlag5(); *setComplete()*;
ComSW!*msg*$_{rep}$*.S.s.W.complete* → *Skip*
else
Store$_{upload}$*(s)*

3.2. Kiểm chứng kiến trúc Haystack

3.2.1. Cài đặt trong PAT

Đầu tiên, chúng tôi xác định các kênh quan trọng, các loại cờ thông điệp, các đối tượng phân phối dưới dạng liệt kê và xác định thông điệp được truyền giữa các kênh dưới dạng các biến toàn cục. Đồng thời, chúng tôi đưa ra biến toàn cục *flag* trong một phần của mã chương trình để đánh giá việc thực thi chương trình, hỗ trợ chương trình hoạt động tốt và tăng cường khả năng đọc. Danh sách định nghĩa của các biến ở trên như sau:

channel ComBW 0;
enum {*msg*$_{req}$, *msg*$_{rep}$, *msg*$_{data}$};
enum {*W, N, C, D, B, S*};
var req_serv;
var none = - 1;
var capacity = 1000;
var state;
#define T 3;
#define M 7;
#define M1 3;
#define M2 2;
#define M3 2;

Tất cả các kênh khác trong mô hình được xác định theo cú pháp định dạng kênh ở trên như *ComBW* (kênh truyền thông giữa Browser và WebServer); các kiểu liệt kê là kiểu thông điệp cờ, bao gồm *msg*$_{req}$ đại diện cho yêu cầu, *msg*$_{rep}$ đại diện cho câu trả lời và *msg*$_{data}$ đại diện cho dữ liệu; *W, N, C, D, B, S* đại diện cho các chữ cái đầu bằng tiếng Anh của sáu mô-đun trong phần mô hình Haystack. Ở đây, chúng tôi đưa ra định nghĩa về một trong các thông điệp trong Bảng 1:

- *req_serv*: biến toàn cục;
- *none*: biến toàn cục được khởi tạo thành -1 nghĩa là không có dữ liệu trả về hoặc hoạt động nào là vô nghĩa;

- *capacity*: biến toàn cục được khởi tạo thành 1000 có nghĩa là kích thước tối đa của máy lưu trữ;
- *state*: biến toàn cục dựa trên hoạt động thực tế của tiến trình Directory theo chức năng tương ứng;

  Ngoài ra, chúng tôi đưa ra định nghĩa của năm biến không đổi, bao gồm *T, M, M1*, *M2* và *M3* có giá trị tương ứng là 3, 7, 3, 2 và 2.

3.2.2. Kiểm chứng các tính chất dựa trên phân tích đại số

Dựa trên việc cài đặt mô hình CSP trong PAT ở trên, chúng tôi thực hiện kiểm chứng các tính chất của kiến trúc Haystack, bao gồm tính chất cơ bản và tính chất bổ sung.

(1) Tính chất cơ bản Deadlock free

#assert System() *deadlockfree*;

Tính chất Deadlock free chủ yếu được sử dụng để mô tả liệu có trạng thái bế tắc cho toàn bộ mô hình CSP hay không. Theo kết quả kiểm chứng của PAT trong Hình 7, việc tóm tắt mô hình CSP ở trên không có bế tắc.

(2) Tính chất bổ sung

Tính chất bổ sung gồm năm tính chất. Đó là truy cập tương tranh đồng bộ, truy cập tương tranh không đồng bộ, truy cập tương tranh đồng bộ với cùng một máy khách, tải nạp tương tranh đồng bộ và tải nạp tương tranh đồng bộ với cùng một máy khách.

(a) Truy cập tương tranh đồng bộ

Synchron() = [*req_serv* == 3](|| *b*:{0..*T*-1}@S*ystem_serv*(*b*,1));

#define goal (*data* == 44);

#assert Synchron() *reaches goal*;

Theo kết quả kiểm chứng, tính chất này là hợp lệ được thể hiện trong Hình 8.

(b) Truy cập tương tranh không đồng bộ

Asynchron() = [*req_serv* == 0]*System_serv*(0,1)
|| [*req_serv* == 3]*System_serv*(1,1)
|| [*req_serv* == 5]*System_serv*(2,1);

#define goal1 (*data* == 11 || *data* == 44 || *data* == 66);

#assert Asynchron() *reaches goal1*;

Theo kết quả kiểm chứng của PAT, tính chất này hợp lệ được thể hiện trong Hình 9.

(c) Truy cập tương tranh đồng bộ với cùng một máy khách

SSynchron() = [*req_serv* == 0]*System_serv*(0,1)
|| [*req_serv* == 3]*System_serv*(0,1)
|| [*req_serv* == 5]*System_serv*(0,1);

#define goal2 (*data* == 11 || *data* == 44 || *data* == 66);

#assert SSynchron() *reaches goal2*;

Theo kết quả kiểm chứng của PAT, tính chất này hợp lệ được thể hiện trong Hình 10.

(d) Tải nạp tương tranh đồng bộ

Synchronup() = [*data* == 88](||| *t*:{0..*T* - 1}@*System_upload*(*t*,1));
#define goal4 complete == 1;
#assert Synchronup() *reaches goal4*;
Hình 11 cho thấy kết quả kiểm chứng của tính chất này là hợp lệ.

(e) Tải nạp tương tranh đồng bộ với cùng một máy khách

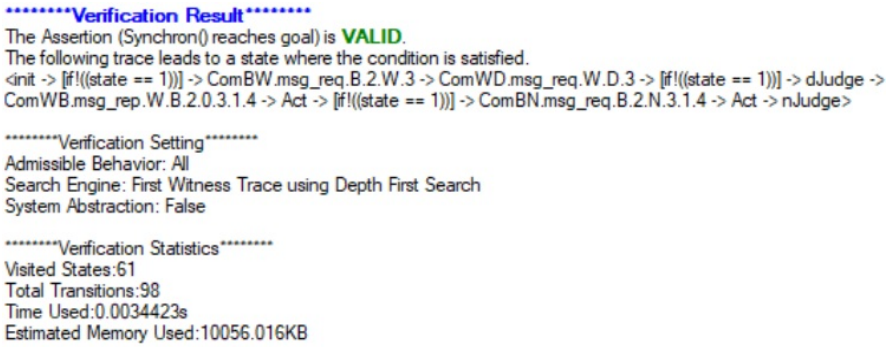
SSynchronup() = [*data* == 99](*System_upload*(0,1)
|| [*data* == 100]*System_upload*(0,1)
|| [*data* == 110]*System_upload*(0,1));
#define goal5 complete == 1;
#assert SSynchronup() *reaches goal5*;
Hình 12 cho thấy kết quả kiểm chứng tính chất này chưa hợp lệ.

3.2.3. Kết quả phân tích và kiểm chứng

Theo kết quả kiểm chứng các tính chất của kiến trúc Haystack bằng PAT, thì tính chất Deadlock free là hợp lệ.

Hình 7. *Kết quả kiểm chứng tính chất Deadlock free*

Tính chất truy cập tương tranh đồng bộ là hợp lệ.



Hình 8. *Kết quả kiểm chứng truy cập tương tranh đồng bộ*

Tính chất truy cập tương tranh không đồng bộ cũng là tính chất hợp lệ.

```
********Verification Result********
The Assertion (Asynchron() reaches goal1) is VALID.
The following trace leads to a state where the condition is satisfied.
<init -> ComBW.msg_req.B.0.W.0 -> ComWD.msg_req.W.D.0 -> dJudge -> [if((state == 1))] ->
ComDW.msg_rep.D.W.1.0.1.1 -> Act -> ComWB.msg_rep.W.B.0.1.0.1.1 -> Act -> [if((state == 1))] ->
ComBC.msg_req.B.0.C.0.1.1 -> Act -> cJudge>

********Verification Setting********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

********Verification Statistics********
Visited States:24
Total Transitions:48
Time Used:0.004116s
Estimated Memory Used:9099.688KB
```

Hình 9. *Kết quả kiểm chứng truy cập tương tranh không đồng bộ*

Tính chất truy cập tương tranh đồng bộ với cùng một máy khách là tính chất hợp lệ.

```
********Verification Result********
The Assertion (SSynchron() reaches goal2) is VALID.
The following trace leads to a state where the condition is satisfied.
<init -> ComBW.msg_req.B.0.W.0 -> ComWD.msg_req.W.D.0 -> dJudge -> [if((state == 1))] ->
ComDW.msg_rep.D.W.1.0.1.1 -> Act -> ComWB.msg_rep.W.B.0.1.0.1.1 -> Act -> [if((state == 1))] ->
ComBC.msg_req.B.0.C.0.1.1 -> Act -> cJudge>

********Verification Setting********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

********Verification Statistics********
Visited States:24
Total Transitions:48
Time Used:0.005248s
Estimated Memory Used:9157.304KB
```

Hình 10. *Kết quả kiểm chứng truy cập tương tranh đồng bộ với cùng một máy khách*

Tính chất tải nạp tương tranh đồng bộ là tính chất hợp lệ.

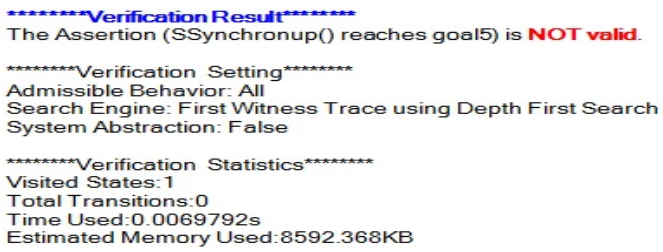
```
********Verification Result********
The Assertion (Synchronup() reaches goal4) is VALID.
The following trace leads to a state where the condition is satisfied.
<init -> ComBW.msg_data.B.2.W.88 -> ComWD.msg_req.W.D.88.0.0.0.0 -> [if(((write_enabled == 1)
&& (num == 7)))] -> [if(((write_enabled == 1) && (num == 9)))] -> Act -> compute -> alloc ->
[if((write_enabled == 1))] -> setFlag7 -> ComBW.msg_data.B.1.W.88 ->
ComWD.msg_req.W.D.88.1.8.2.8 -> Act -> compute -> alloc -> [if((write_enabled == 1))] -> setFlag7 ->
[if(((write_enabled == 1) && (num == 7)))] -> ComDW.msg_rep.D.W.88.1.8.2.8 -> Act -> setcookie ->
setFlag8 -> [if(((write_enabled == 1) && (num == 8)))] -> ComWS.msg_data.W.S.1.88.101 -> Act ->
setFlag9 -> setCom>

********Verification Setting********
Admissible Behavior: All
Search Engine: First Witness Trace using Depth First Search
System Abstraction: False

********Verification Statistics********
Visited States:100
Total Transitions:163
Time Used:0.0158434s
Estimated Memory Used:9892.288KB
```

Hình 11. *Kết quả kiểm chứng tải nạp tương tranh đồng bộ*

Cuối cùng, tính chất tải nạp tương tranh đồng bộ với cùng một máy khách là không hợp lệ.



Hình 12. *Kết quả kiểm chứng tải nạp tương tranh đồng bộ với cùng một máy khách*

4. Kết luận

Kiến trúc Haystack là hệ thống lưu trữ ảnh của Facebook. Sau khi nghiên cứu, bài viết đã phân tích được các thành phần Browser, WebServer, Directory, CDN, Cache và Store trong kiến trúc Haystack theo quan điểm của đại số tiến trình với CSP. Chúng tôi cũng đã áp dụng bộ công cụ phân tích tiến trình PAT để cài đặt và kiểm chứng mô hình đã xây dựng và các tính chất của kiến trúc Haystack. Ngoài ra, chúng tôi đã kiểm chứng được một tính chất cơ bản và năm tính chất bổ sung được tóm tắt từ tài liệu của kiến trúc Haystack, bao gồm Deadlock free, truy cập tương tranh đồng bộ, truy cập tương tranh không đồng bộ, truy cập tương tranh đồng bộ với cùng một máy khách, tải nạp tương tranh đồng bộ và tải nạp tương tranh đồng bộ với cùng một máy khách. Các tính chất hợp lệ ngoại trừ tính chất cuối cùng. Vì vậy, chúng tôi có thể nói rằng từ góc độ của đại số tiến trình CSP, mô hình được xây dựng đáp ứng các tính chất này. Nói cách khác, kiến trúc Haystack đáp ứng nhu cầu về tài liệu của nó.

Trên thực tế, chúng tôi nhận thấy rằng nó vẫn là một thách thức lớn để lập mô hình và kiểm chứng kiến trúc Haystack. Trong tương lai, chúng tôi sẽ tiến hành một số nghiên cứu về phân tích bảo mật xác thực danh tính trong kiến trúc Haystack.

❖ ***Tuyên bố về quyền lợi:*** *Các tác giả xác nhận hoàn toàn không có xung đột về quyền lợi.*

TÀI LIỆU THAM KHẢO

Brookes, S. D., Hoare, C. A. R., & Roscoe, A. W. (1984). A theory of communicating sequential processes. *J. ACM*, *31*(3), 560-599.

Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, & Peter Vajgel (2010). Finding a needle in haystack: Facebook's photo storage. *In 9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, 47-60.

Hoare, C. A. R. (1978). Communicating sequential processes. *Communications of the ACM*, *21*(8), 666-677.

Jan A. Bergstra, & Jan Willem Klop (1985). Algebra of communicating processes with abstraction. *Theor. Comput. Sci.*, 37, 77-121.

Lowe, G., & Roscoe A. W. (1997). Using CSP to detect errors in the TMN protocol. *IEEE Trans. Software Eng., 23*(10), 659-669.

National University of Singapore. PAT (2008). Process analysis toolkit url=https://pat.comp.nus.edu.sg/.

Ngo, Q. V. (2011). Multi-coronas zernike moments on curvelet-like transform and application to pattern recognition. *Ho Chi Minh City University of Education of Journal of Science,* 30(64).

Robin Milner (1980) *A Calculus of Communicating Systems, volume 92 of Lecture Notes in Computer Science*. Springer.

Roscoe, A. W. (2010). *Understanding Concurrent Systems*. Texts in Computer Science. Springer.

ANALYSIS AND FORMALIZATION OF HAYSTACK ARCHITECTURE IN FACEBOOK

Le Thi Thuy, Bui Quoc Viet*

Ho Chi Minh City University of Physical Education and Sport, Vietnam

**Corresponding author: Bui Quoc Viet – Email: vietqb@upes.edu.vn*

Received: October 11, 2022; Revised: November 05, 2022; Accepted: June 26, 2023

ABSTRACT

Haystack is a storage system architecture optimized for Facebook's photo application. Current haystack has four main advantages compared to previous versions, including high throughput and low latency, fault tolerance, cost effectiveness, and simplicity. Given the widespread use of the Haystack architecture in Facebook, its validity and other key attributes abstracted from this architecture need to be analyzed and verified with a more precise approach. This paperfocuses on the internal design of serving and uploading a photo of Haystack architecture and apply Communicating Sequential Processes (CSP) to formalize them in detail. By feeding the models into the model checker Process Analysis Toolkit (PAT), we have verified crucial properties, including basic property and supplementary properties. Basic property contains Deadlock Freedom. Supplementary properties include synchronous and asynchronous concurrent access, and synchronous concurrent access with the same client, synchronous concurrent and synchronous concurrent upload with the same client. Finally, according to the verification results, we believe that from the CSP's perspective, the properties of Haystack architecture are valid, which means that it meets the requirements of the documents of Facebook.

Keywords: analysis; CSP; Haystack, formalization; PAT