A. Giới thiệu  
Hệ phân tán đóng vai trò quan trọng trong lĩnh vực phát triển của công nghệ máy  
tính, đăc biệt là trong điều kiện phát triển bùng nổ của các mạng máy tính. Sự phát triển  
của các mạng LAN, WAN cho phép hàng trăm, hàng nghìn hay thậm chí hàng triệu mát  
tính có thể kết nối với nhau.  
Kết quả của sự phát triển công nghệ hiện nay không chỉ đáng tin cậy mà còn tạo nên các  
hệ thống máy tính rất lớn, được kết nối bằng các đường kết nối tốc độ cao. Chúng tạo nên  
các mạng máy tính lớn hoặc các hệ phân tán, ngược với hệ thống tập trung trước đây, bao  
gồm các máy tính đơn và có thể cả thiết bị điều khiển đầu cuối (remote teminal).  
Có nhiều định nghĩa được đưa ra, nhưng nếu coi hệ phân tán là hệ thống phục vụ người  
dùng thì : hệ phân tán là tập các máy tính độc lập giao tiếp với người dùng như một hệ  
thống thống nhất và trọn vẹn.  
Hệ phân tán được xây dựng cần đảm bảo một số đặc trưng :  
¬ Chia sẻ tài nguyên  
¬ Tính trong suốt  
¬ Tính mở  
¬ Tính co giãn  
Tính trong suốt đảm bảo khả năng che giấu tiến trình và tài nguyên phân tán trong mạng  
máy tính. Hệ phân tán có khả năng biểu diễn bản thân nó với người dùng và ứng dụng  
giống như một máy tính đơn lẻ.  
Bài tiểu luận này sẽ trình bày về các đặc trưng về mặt lý thuyết để đảm bảo tính trong suốt trong hệ phân tán và triển khai nó trong hệ thống WEB.

B. Nội dung  
**I. Các vấn đề về lý thuyết của hệ phân tán**

**1. Tính trong suốt đối với người sử dụng**

• Truy nhập (Access transparency): che giấu sự khác biệt trong cách biểu hiện dữ  
liệu và cách thức truy nhập tài nguyên. Ở mức cơ bản, ta che giấu sự khác biệt về  
kiến trúc máy, nhưng quan trọng hơn chính là ta phải đạt được sự thống nhất trong  
biểu diễn dữ liệu bởi các máy tính khác nhau và các hệ điều hành khác nhau. Ví  
dụ, quy ước về cách đặt tên của các máy tính khác nhau chạy trên các hệ điều  
hành khác nhau là khác nhau, tuy nhiên cách thức này cùng với các thao tác với  
file hoàn toàn trong suốt với cả ứng dụng và người dùng.  
• Vị trí (Location transparency): che giấu vị trí tài nguyên → người dùng hoàn toàn  
không biết về vị trí vật lý của tài nguyên trong hệ thống. Để đạt được , ta cần tiến  
hành định danh bằng tên gọi logic. Tên gọi có thể đơn giản, không cần mã hóa.  
VD : cardchua.vn, là tên của trang web, ta không hề biết vị trí vật lý của nó trên  
Web server nào, nhưng vẫn có thể truy cập được.  
• Di trú (Migration transparency): che giấu việc tài nguyên di chuyển sang máy  
khác → tài nguyên có thể không nằm ở vị trí cố định, mà nó có thể được di  
chuyển sang các máy khác nhau trong hệ thống để phục vụ các yêu cầu khác nhau  
mà không ảnh hưởng tới việc truy nhập tài nguyên.  
• Di chuyển (Relocation transparency): che giấu việc tài nguyên có thể bị di chuyển  
sang nơi khác → tài nguyên có thể di chuyển sang nơi khác ngay cả khi nó đang  
bị truy cập. VD: người dùng điện thoại di động có thể di chuyển từ nơi này sang  
nơi khác, do đó chuyển liên lạc từ trạm phát sóng này sang trạm phát sóng khác  
nhưng cuộc thoại vẫn liên tục, không có bất kì gián đoạn nào.  
• Nhân bản (Repilcation transparency): che giấu việc sao chép tài nguyên → việc  
sao chép tài nguyên giúp đơn giản hóa và tăng tốc độ truy cập dữ liệu, các bản sao  
đặt gần hoặc ngay tại nơi truy cập dữ liệu. Các bản sao này phải có cùng tên, để  
che giấu với người dùng; và hệ thống cần có tính trong suốt về vị trí, để đảm bảo  
khả năng quản lý các bản sao khác nhau trên các máy trong hệ thống.  
• Tương tranh (Concurrency transparency):

che giấu sự chia sẻ tài nguyên bởi một  
số người sử dụng, nhiều người dùng có thể cùng truy cập dữ liệu tại cùng một thời  
điểm, đặc biệt sử dụng nhiều trong mạng truyền thông. VD: hai người cùng lưu  
trữ file trên cùng server, và tại một thời điểm, họ truy cập tới cùng bản ghi trong  
cơ sở dữ liệu chung, để đảm bảo tính nhất quán và ổn định của dữ liệu với từng  
người dùng, ta cần sử dụng các cơ chế khóa. Ngoài ra, còn có thể thực hiện cơ chế  
giao dịch, tuy nhiên nó khó thực hiện trên hệ thống phân tán.  
• Lỗi (Failure transparency): che giấu lỗi và khắc phục lỗi → đảm bảo người dùng  
hoàn toàn không biết về các lỗi xảy ra trong hệ thống và sự khắc phục các lỗi này.  
Che dấu lỗi là một trong các yêu cầu khó thực hiện nhất, và có thể không thực  
hiện được trong một số tình huống cụ thể. Điểm khó nhất chính là ta không thể  
phân biệt được tài nguyên là truy cập chậm hay không thể truy cập được. VD: khi  
truy cập một web Server bận, trình duyệt thông báo “time-out”, thì ta không thể  
biết được server có thực sự bị lỗi hay không.  
Mức độ trong suốt  
Tuy nhiên không phải lúc nào cũng nên thể hiện tính trong suốt hoàn toàn với người  
dùng. Ví dụ, khi có báo điện tử xuất hiện trong hòm thư của người dùng vào lúc 7h, trong  
khi bạn đang ở múi giờ khác, người dùng cần phải biết thông tin về tờ báo này không  
phải là tờ báo mà họ vẫn sử dụng. Ngoài ra cũng có sự đánh đổi giữa mức độ trong suốt  
và tốc độ hệ thống. Ví dụ, nhiều ứng dụng Internet lặp lại việc truy cập máy chủ sau một  
khoảng thời gian nhất định trước khi từ bỏ, do đó, nếu ta cố che dấu việc kết nối thất bại  
sẽ làm chậm đáng kể tốc độ hệ thống. Với trường hợp này, cần rút ngắn thời gian cố kết  
nối hoặc để cho người dùng huỷ bỏ kết nối.  
Ví dụ khác, ta cần đảm bảo sự thống nhất giữa các bản sao trên các máy khác nhau. Một  
bản ghi bị thay đổi sẽ kéo theo sự thay đổi của tất cả các bản ghi khác, làm chậm đáng kể  
thời gian hệ thống, nên không thể che giấu người dùng.  
Trong một số tình huống, tính trong suốt không hẳn là giải pháp tốt, cụ thể như trong các  
tình huống liên quan tới vị trí và hoàn cảnh đặc biệt, chúng ta nên thể hiện tính phân tán  
của hệ thống hơn là nên che dấu nó, cụ thể như trong các hệ phân tán nhúng và phân bố rộng. Ví dụ, khi ta muốn in văn bản thông qua các máy in mạng, nên gửi tới các máy in ở  
gần ta, dù đang bận, hơn là tại các máy in xa.  
Việc thiết kế hệ phân tán trong suốt là cần thiết, tuy nhiên cũng cần đảm bảo sự hài hoà  
với các đặc tính khác như hiệu năng và tính thân thiện với người dùng của hệ thống. Tuy  
nhiên có thể phải trả giá đắt cho việc không đảm bảo tính trong suốt hoàn toàn của hệ  
phân tán.

**2. Kiến trúc**

Hệ phân tán thường bao gồm tập phức tạp của các phần mềm nằm rải rác trên các máy  
khác nhau, do đó cần tổ chức tốt. Có nhiều cách khác nhau để xem xét tổ chức của hệ  
phân tán , và một trong số đó là tách riêng tổ chức logic của các thành phần phần mềm và  
các phần cứng hệ thống.  
Một trong các mục đích quan trọng của hệ phân tán là phân tách các ứng dụng từ nền bên  
dưới thông qua lớp trung gian. Đó chính là bước thiết kế quan trọng với mục đích chính  
là đảm bảo tính trong suốt phân tán. Tuy nhiên, việc đó phải đánh đổi bằng nhiều biện  
pháp thiết kế phức tạp để làm lớp trung gian có khả năng tương tác tốt.  
Các loại kiến trúc hệ thống:  
- kiến trúc lớp  
- kiến trúc hướng đối tượng  
- kiến trúc dữ liệu trung tâm  
- kiến trúc hướng sự kiện  
Kiến trúc lớp : các thành phần được tổ chức theo kiểu lớp, thành phần lớp L(i)  
sẽ gọi thành  
phần ở lớp dưới L(i-1)  
, mô hình được sử dụng rộng rãi trong truyền thông mạng .  
Kiến trúc hướng đối tượng có các đối tượng được định nghĩa dưới dạng các thành phần  
và được kết nối với nhau thông qua cơ chế gọi thủ tục (remote). Dạng kiến trúc phần  
mềm này tương tự như kiến trúc client-server.  
Hai dạng kiến trúc lớp và hướng đối tượng vẫn đóng vai trò quan trọng trong các hệ  
thống phần mềm lớn.  
Kiến trúc dữ liệu tập trung suy ra từ ý tưởng các tiến trình giao tiếp thông qua phần tử  
(chủ động hay thụ động) chung. Kiến trúc này cũng quan trọng không kém 2 kiến trúc  
trên, ví dụ như hệ thống chia sẻ file chung nhằm che giấu tất cả các kết nối trong hệ  
thống thông qua các liên kết ảo tới file như trong hệ thống phân tán kiểu Web.  
Trong kiến trúc hướng sự kiện, tiến trình về cơ bản sẽ giao tiếp thông qua sự lan truyền  
của sự kiện, có thể mang cả dữ liệu. Với hệ phân tán, sự lan truyền sự kiện nói chung liên  
quan tới các hệ thống công cộng (publish/subscribe). Ý tưởng cơ bản là các tiến trình gửi  
sự kiện sau khi lớp trung gian đảm bảo chỉ các tiến trình chấp nhận sự kiện này mới có  
thể nhận chúng. Lợi ích lớn của hệ thống hướng sự kiện là các tiến trình có mỗi quan hệ  
linh động, chúng không cần liên quan chặ chẽ với các tiến trình khác. Do đó chúng có thể  
liên kết hoặc tự do.  
Kiến trúc hướng sự kiện có thể kết hợp với kiến trúc dữ liệu tập trung, tạo thành không  
gian dữ liệu chia sẻ. Bản chất của không gian dữ liệu tập trung là các tiến trình có thể  
không liên kết với nhau, chúng không cần sẵn sàng khi sự truyền đạt được thiết lập. Xa  
hơn, nhiều không gian dữ liệu chia sẻ sử dụng giao diện SQL để chia sẻ kho dữ liệu với ý  
nghĩa dữ liệu có thể truy nhập thông qua mô tả hơn là tham chiếu rõ ràng, như trong  
trường hợp với file.  
Sự cần thiết của các kiến trúc phần mềm để đảm bảo tính trong suốt đối với các hệ phân  
tán là không có gì phải bàn cãi, tuy nhiên yêu cầu về tính trong suốt phải đánh đổi với  
hiệu năng hệ thống, khả năng chịu lỗi, khả thi khi lập trình và còn một số các yêu cầu  
khác. Không có lời giải tách biệt nào đảm bảo tất cả yêu cầu của hệ phân tán, do đó ta cần  
áp dụng tổng hợp các giải pháp để đạt hiệu quả.

**3. Tiến trình**

Khái niệm của tiến trình bắt nguồn từ các lĩnh vực của hệ điều hành , trong đó nó được  
định nghĩa như là chương trình đang thực thi . Trong đó thì việc quản lý và lập kế hoạch  
của các tiến trình có lẽ là vấn đề quan trọng nhất cần phải quan tâm .  
Ví dụ , để tổ chức hiệu quả 1 hệ thống client-server, người ta thường sử dụng các kỹ  
thuật đa luồng. Như chúng ta đã thảo luận trong phần trước, đặc điểm quan trọng nhất  
của luồng trong hệ phân tán là chúng cho phép các máy trạm và máy chủ có thể được xây  
dựng để các kết nối và xử lý nội bộ có thể chồng lên nhau , điều này dẫn đến hiệu suất  
cao , Trong những năm gần đây , khái niệm về ảo hóa đã trở nên phổ biến . Ảo hóa cho  
phép một ứng dụng và có thể có môi trường hoàn chỉnh bao gồm hệ điều hành , để chạy  
đồng thời với các ứng dụng khác nhưng hoàn toàn độc lập về phần cứng và nền tảng nằm  
5  
dưới . Hơn nữa , ảo hóa giúp tách được các sự cố gây ra do lỗi hay các vấn đề về bảo mật.  
Đây là một khái niệm quan trọng trong hệ phân tán .  
Một vấn đề quan trọng , đặc biệt là ở khu vực phân tán rộng, là việc di chuyển tiến tình  
giữa các máy tính khác nhau . Quá trình di chuyển hay cụ thể hơn là di trú mã.

3.1. Luồng

Để thực thi 1 chương trình , hệ điều hành tạo ra một số bộ xử lý ảo, mỗi một bộ xử lý này  
chạy 1 chương trình khác nhau. Để theo dõi các bộ xử lý ảo này , hệ điều hành có 1 bảng  
tiến trình, mỗi mục lưu các giá trị thanh ghi, ánh xạ bộ nhớ, các tập tin mở, các thông tin  
tính toán, quyền hạn … Một tiến trình thường được định nghĩa như là một chương trình  
đang thực thi. Một vấn đề quan trọng là hệ điều hành sẽ theo dõi chặt chẽ để đảm bảo  
rằng các tiến trình độc lập không vô tình hoặc cố ý ảnh hưởng đến các hành vi của tiến  
trình khá. Tuy nhiên, thực tế là nhiều quá trình có thể chia sẻ cùng một bộ xử lý và tài  
nguyên phân cứng khác do tính trong suốt của hệ thống. Thông thường , hệ điều hành yêu  
cầu phần cứng hỗ trợ để thực hiện việc phân chia này.  
Điều này làm chi phí cho các tiến trình sẽ rất lớn . Ví dụ , mỗi lần 1 tiến trình được tạo ra,  
hệ điều hành cần phải tạo ra một không gian địa chỉ độc lập. Phân phối bộ nhớ được xem  
như khởi tạo phân đoạn bộ nhớ bằng cách xoá dữ liệu, sao chép chương trình vào phân  
đoạn text, và khởi tạo ngăn xếp . Tương tự, chuyển đổi phục vụ của CPU giữa 2 tiến trình  
có thể tương đối tốn kém. Ngoài việc tiết kiệm CPU ( các giá trị thanh ghi , bộ đếm  
chương trình , ngăn xếp con trỏ ) hệ điều hành cũng sẽ phải sửa đổi thanh ghi của bộ  
nhớ quản lý (MMU ) và bộ nhớ cache như trong bộ đệm TLB. Thêm vào đó, nếu hệ  
thống hỗ trợ nhiều tiến trình đồng thời nó cần phải lưu tiến trình trong bộ nhớ chính. Nó  
có thể có tiến trình nháp để trao đổi giữa bộ nhớ chính và ổ đĩa trước khi thực sự thực  
hiện tiến trình.  
Giống như tiến trình, một luồng thực thi các đoạn mã của nó độc lập với các luồng khác.  
Tuy nhiên, ngược với tiến trình nó không cố gắng để tạo ra sự độc lập với các luồng khác  
nếu việc này làm giảm hiệu năng. Do đó một luồng hệ thống thường chỉ duy trì các thông  
tin tối thiểu cho phép một CPU được chia sẻ bởi 1 vài luồng. Đặc biệt, một bối cảnh  
luồng thông thường chỉ gồm có bối cảnh CPU cùng với một số thông tin khác để quản lý  
luồng. Ví dụ , một hệ thống luồng có thể đảm bảo rằng một luồng hiện tại đang bị khóa  
thì sẽ không thể thực thi. Thông tin đó không thực sự cần thiết để quản lý đa luồng và  
6  
thường bị bỏ qua. Vì lý do này , bảo vệ dữ liệu chống lại các truy cập không thích hợp  
bằng luồng trong 1 tiến trình được đặt hoàn toàn vào người phát triển ứng dụng .  
Có hai ý nghĩa quan trọng của phương pháp này. Trước hết , hiệu suất của 1 ứng dụng đa  
luồng khó có thể kém hơn ứng dụng đơn luồng. Trong thực tế, nhiều trường hợp đa luồng  
làm cho hiệu suất tăng . Thứ hai, bởi vì các luồng không tự động bảo vệ lẫn nhau như  
cách của tiến trình, nên việc phát triển các ứng dụng đa luồng cần được phát triển hơn  
nữa, đảm bảo thiết kế và giữ mọi thứ càng đơn giản càng tốt. Tuy nhiên, những thực  
nghiệm hiện nay không chứng mình rằng nguyên tắc này đều được các nhà phát triển  
nắm rõ.

3.2. Di trú mã

Di trú mã là một trong các đặc tính của tính trong suốt trong hệ phân tán, nhằm di trú tiến  
trình (ngay cả khi chúng đang thực thi), nhằm đơn giản hoá việc thiết kế hệ phân tán.  
Phương pháp tiếp cận  
Di trú mã trong hệ phân tán là quá trình di chuyển toàn bộ tiến trình từ máy này sang máy  
khác. Tuy điều này làm tốn kém và phức tạp nhưng hiệu quả mang lại về hiệu năng là rõ  
rệt. Ý tưởng cơ bản là hiệu năng hệ thống có thể tăng rõ rệt nếu các tiến trình được di  
chuyển từ máy nặng tải sang máy nhẹ tải hơn. Các thuật toán phân chia tải cho hệ thống  
liên quan tới phân chia và tái phân chia các tác vụ với tập hợp các vi xử lý, đóng vai trò  
quan trọng trong các hệ thống tính toán cường độ lớn. Mặc dù vậy, trong các hệ thống  
tính toán hiện đại, tối ưu tính toán tải ít được đưa ra hơn. Hơn nữa, do tính không đồng  
nhất của nền bên dưới và mạng máy tính, việc cải thiện hiệu năng thông qua di trú mã  
thường dựa trên lý do định tính hơn là định lượng.  
Ví dụ, trong trường hợp server chứa lượng cơ sở dữ liệu lớn, nếu client cần thực hiện  
nhiều hoạt động trên CSDL này, có thể nên chuyển một phần mã của ứng dụng client  
sang server và chỉ gửi kết quả qua mạng, giúp giảm tải đáng kể. Điều này có thể áp dụng  
tương tự cho phía client.  
Di trú mã có thể được áp dụng trong tính toán song song, dù không rắc rối như lập trình  
song song. Ví dụ cụ thể như với trường hợp tìm kiếm thông tin trên WEB, được thực hện  
thông qua tác tử di động, di chuyển từ site này sang site khác. Bằng cách sao chép tác tử  
này, do kích thước nhỏ nên thời gian thực hiện nhanh, và gửi chúng tới các site khác  
nhau, ta có thể thu dược hiệu quả như sử dụng một chương trình đơn lẻ.  
Bên cạnh việc cải thiện hiệu năng, còn một số lý do khác. Lý do quan trọng nhất là độ  
linh động của hệ thống. đơn giản nhất là ta chia ứng dụng thành các đoạn khác nhau, và  
đánh giá đoạn nào nên được thực thi.Tuy nhiên, nếu mã di chuyển giữa các máy khác  
nhau, ta có thể cấu hình động cho hệ phân tán.  
Mô hình di trú mã  
Mặc dù di trú mã có nghĩa là ta chỉ chuyển mã giữa các máy, điểu này bao hàm khu vực  
rộng lớn hơn nhiều. Theo truyền thống, truyền thông trong hệ phân tán liên quan tới việc  
trao đổi dữ liệu giữa các tiến trình. Di trú mã theo nghĩa rộng là di chuyển các chương  
trình giữa các máy với nhau, với mục tiêu thực hiện các chương trình này để đạt được  
mục đích. Trong một số trường hợp, trạng thái thực thi chương trình, thông tin hiện hành  
và một số phần khác của môi trường cũng được di chuyển theo.  
Ta sử dụng mô hình sau : trong mô hình này, tiến trình được chia thành 3 phần. Phần mã  
chứa tập lệnh thực hiện của tiến trình. Phần tài nguyên chứa các tài nguyên cần thiết như  
file, máy in, các thiết bị và các tiến trình khác. Cuối cùng, phấn thực thi lưu giữ các trạng  
thái của tiến trình trong thời điểm hiện tại, gồm dữ liệu riêng, ngăn xếp và bộ đếm  
chương trình.  
Ta có 2 mô hình chính của di trú mã: mô hình di động yếu và mô hình di động mạnh.  
Trong mô hình di động yếu, ta có thể chỉ chuyển phần mã, với một số dữ liệu ban đầu.  
Chức năng chính của mô hình này là chương trình được truyền luôn bắt đầu từ một trong  
các vị trí bắt đầu được xác định trước. Lợi ích rõ ràng nhất của phương pháp này là sự  
đơn giản, chỉ yêu cầu máy đích thực thi mã, thực tế làm giảm sự linh động của mã.  
Ngược với mô hình di động yếu, trong mô hình di động mạnh, tiến trình đang chạy có thể  
dừng lại và tiếp tục trạng thái đang hoạt động khi di chuyển từ máy này sang máy khác.  
Tuy nhiên nó phải trả giá bằng việc khó triển khai hơn so với mô hình di động yếu.  
Trong trường hợp mô hình di động yếu, có sự khác nhau nếu mã di trú được thực thi bởi  
tiến trình đích và với các tiến trình riêng biệt, ví dụ với Java applet được tải về bởi trình  
duyệt được thực thi trong không gian địa chỉ của trình duyệt. Lợi ích rõ ràng là không cần  
kết nối tới máy chủ, tuy nhiên cần bảo vệ tiến trình đích trước mã độc và sự sơ xuất khi  
thực thi. Có thể giải quyết bằng cách hệ điều hành chú ý tới việc tạo các tiến trình để thực  
thi mã di trú. Tuy nhiên nó không giải quyết được vấn đề truy nhập tài nguyên trái phép.  
Thay vì di chuyển các tiến trình đang chạy, mô hình di động mạnh được hỗ trọ bởi các  
remote cloning. Trái ngược với di trú mã, ta sao chép hoàn toàn chính xác tiến trình gốc,  
nhưng được thực hiện trên máy khác. Trong hệ thống UNIX, remote cloning chia nhánh  
một tiến trình con và để tiến trình này tiếp tục thực thi trên máy điều khiển. Lợi ích của  
việc nhân bản là tạo mô hình gần như tập hợp bởi các ứng dụng khác nhau đã được sử  
dụng. Điểm khác duy nhất là tiến trình nhân bản được thực thi trên máy khác. Do đó, việc  
di trú bằng cách nhân bản là cách đơn giản để cải thiện tính trong suốt.  
Tài nguyên di trú và tài nguyên cục bộ  
Cho tới phần này, ta mới chỉ xét tới mã di trú và các phần thực thi (execution segment)  
được thực hiện … Phân đoạn tài nguyên đòi hỏi những sự chú ý đặc biệt. Điều khó khăn  
thường gặp khi thực hiện di trú mã đó là phân đoạn tài nguyên không phải luôn dễ dàng  
được chuyển đi cùng các phân đoạn khác mà không bị thay đổi. Ví dụ, giả sử rằng một  
tiến trình nắm giữ một tham chiếu tới một cổng TCP mà qua đó nó giao tiếp với các tiến  
trình khác. Do một tham chiếu được giữ trong phân đoạn tài nguyên của nó, khi tiến trình  
di chuyển sang vị trí khác, nó sẽ phải từ bỏ cổng cũ và yêu cầu một cổng mới ở vị trí mới.  
Để hiểu sự hàm ý mà di trú mã có trong phân đoạn tài nguyên, ta phân biệt 3 loại ràng  
buộc tiến trình-tài nguyên. Loại ràng buộc mạnh nhất là khi một tiến trình đòi hỏi tài  
nguyên dựa vào định danh của nó. Trong trường hợp đó, tiến trình sẽ yêu cầu chính xác  
tài nguyên tham chiếu, ngoài ra không còn yêu cầu gì khác. Một ví dụ của ràng buộc bởi  
định danh là khi một tiến trình sử dụng một VRL để yêu cầu một Web site cụ thể nào đó  
hoặc yêu cầu tới một FrP server bởi địa chỉ Internet của server đó. Với cùng nguyên nhân  
đó, những tham chiếu tới các điểm giao tiếp cục bộ cũng dẫn tới ràng buộc bởi định danh.  
Một dạng ràng buộc tiến trình – tài nguyên yếu hơn là khi ta chỉ quan tâm tới giá trị của  
tài nguyên được yêu cầu. Trong trường hợp đó, sự thực thi của tiến trình sẽ không bị ảnh  
hưởng nếu các tài nguyên khác cũng cung cấp cùng một giá trị. Một ví dụ cho ràng buộc  
bởi giá trị là khi một chương trình dựa trên các thư viện chuẩn, như lập trình C hoặc Java  
chẳng hạn. Các thư viện này luôn tồn tại ở cục bộ, nhưng vị trí chính xác của chúng trong  
hệ thống file cục bộ có thể khác nhau giữa các site khác nhau. Nội dung của chúng rất  
quan trọng cho việc thực thi chính xác của các tiến trình.  
Cuối cùng, dạng ràng buộc yếu nhất là khi một tiến trình chỉ yêu cầu một loại nhất định  
nào đó của tài nguyên.Ràng buộc bởi loại đơn giản nhất đó là tham chiếu tới các thiết bị  
cục bộ, như màn hình, máy in,…  
Khi di trú mã, chúng ta thường cần thay đổi tham chiếu tới các tài nguyên, nhưng không  
thể làm ảnh hưởng tới loại ràng buộc tiến trình – tài nguyên. Một tham chiếu được thay  
đổi chính xác như thế nào, phụ thuộc vào tài nguyên được di chuyển cùng với mã tới máy  
đích. Cụ thể hơn, chúng ta cần quan tâm tới ràng buộc tài nguyên – máy, và phân biệt các  
trường hợp khác nhau.Tài nguyên tách rời có thể dễ dàng di chuyển giữa các máy khác  
nhau, và là các file cụ thể chỉ liên quan đến chương trình được di trú. Ngược lại, di  
chuyển hay sao chép một tài nguyên gắn kết là có thể, nhưng phải tốn chi phí rất cao. Ví  
dụ, về tài nguyên gắn kết đó là các cơ sở dữ liệu cục bộ và những Website hoàn chỉnh.  
Mặc dù các tài nguyên, xét về lý thuyết, không phụ thuộc vào máy chứa nó, nhưng  
thường là không thể di chuyển chúng tới môi trường khác. Cuối cùng, tài nguyên cố định  
được gắn kết cố định với một máy tính hoặc môi trường cụ thể và không thể bị di chuyển.  
Tài nguyên cố định thường là các thiết bị cục bộ. Một ví dụ khác của tài nguyên cố định  
đó là các điểm truyền thông cục bộ.  
Kết hợp 3 loại ràng buộc tiến trình – tài nguyên, và 3 loại ràng buộc tài nguyên – máy  
tính, dẫn tới 9 kết hợp ta cần quan tâm khi thực hiện di trú mã. 9 kiểu kết hợp này được  
chỉ ra trong hình  
  
Trước hết chúng ta quan tâm tới khả năng một tiến trình được gắn với tài nguyên bởi định  
danh. Khi tài nguyên bị tách rời, điều tốt nhất là di chuyển nó cùng với mã di trú. Tuy  
nhiên, khi tài nguyên được chia sẻ bởi các tiến trình khác, một lựa chọn khác là thiết lập  
một tham chiếu toàn cục, một tham chiếu có thể sử dụng ngoài giới hạn của một máy đơn  
lẻ. Một ví dụ của một tham chiếu toàn cục đó là một URL. Khi tài nguyên được gắn kết  
hoặc cố định, giải pháp tốt nhất đó là tạo ra một tham chiếu toàn cục.  
Một điều quan trọng cần nhận ra là việc thiết lập một tham chiếu toàn cục phức tạp hơn là  
việc sử dụng URLs, và việc sử dụng một tham chiếu toàn cục đôi khi rất tốn kém. Ta hãy  
ví dụ, một chương trình tạo ảnh chất lượng cao cho một máy trạm đa phương tiện chuyên  
dụng. Việc tạo ảnh chất lượng cao trong thời gian thực là một nhiệm vụ đòi hỏi tính toán  
rất lớn, cho nên chương trình đó nên được chuyển tới một máy chủ tính toán hiệu năng  
cao. Thiết lập một tham chiếu toàn cục tới một máy trạm đa phương tiện nghĩa là thiết lập  
một đường truyền thông giữa máy chủ tính toán và máy trạm. Thêm vào đó, còn có quá  
trình xử lý tín hiệu ở đồng thời cả máy chủ và máy trạm để đạt tới yêu cầu về băng thông  
cho truyền tải ảnh. Kết quả là việc chuyển chương trình sang máy chủ tính toán không  
phải là một ý hay, chỉ bởi vì chi phí của tham chiếu toàn cục là quá cao.  
Một ví dụ khác cho việc không phải lúc nào cũng có thể thiết lập tham chiếu toàn cục  
một cách dễ dàng khi di trú một tiến trình có sử dụng một điểm truyền thông cuối cục bộ.  
Trong trường hợp đó, ta phải xử lý một tài nguyên cố định mà theo đó tiến trình được  
giới hạn bởi định danh. Có hai giải pháp cơ bản. Một giải pháp đó là để cho tiến trình  
thiết lập một kết nối tới máy tài nguyên sau khi nó được di trú và cài đặt một tiến trình  
riêng biệt ở máy tài nguyên để làm nhiệm vụ đơn giản là chuyến tiếp tất cả các thông  
điệp gửi đến. Điểm hạn chế chính của cách này là nếu như máy tính tài nguyên gặp sự cố  
thì việc giao tiếp với tiến trình di trú sẽ bị thất bại. Giải pháp thứ hai đó là cho tất cả các  
  
tiến trình có giao tiếp với tiến trình di trú thay đổi tham chiếu toàn cục của chúng, và sau  
đó gửi thông điệp tới điểm cuối truyền thông mới trên máy đích.  
Tình hình sẽ khác đi trong trường hợp xử lý ràng buộc bởi giá trị. Trước hết hãy xét một  
tài nguyên cố định. Sự kết hợp của một tài nguyên cố định và ràng buộc bởi giá trị xảy ra,  
ví dụ khi một tiến trình chiếm bộ nhớ được chia sẻ giữa các tiến trình. Thiết lập một tham  
chiếu toàn cục trong trường hợp này nghĩa là chúng ta cần thiết lập một dạng phân tán  
của bộ nhớ chia sẻ. Trong nhiều trường hợp, đây không thực sự là một giải pháp khả thi.  
Tài nguyên gắn kết thường được quy với giá trị của chúng, thường là các thư viện  
runtime. Thông thường, các bản sao của các tài nguyên đó đã có ở trên máy đich, hoặc  
nên được sao chép trước khi di trú mã được thực hiện. Thiết lập một tham chiếu toàn cục  
là cách tốt hơn khi lượng lớn dữ liệu được sao chép, như trường hợp từ điển trong các hệ  
thống xử lý văn bản.  
Trường hợp dễ nhất là khi xử lý với các tài nguyên tách rời. Giải pháp tốt nhất đó là sao  
chép (hoặc di chuyển) tài nguyên tới đích mới, trừ khi nó được chia sẻ bởi một số tiến  
trình. Trong các trường hợp khác, thiết lập một tham chiếu toàn cục là lựa chọn duy nhất.  
Trường hợp cuối cùng là xử lý ràng buộc bởi loại. Không kể đến ràng buộc tài nguyên –  
máy tính, các giải pháp trước đó cố tái liên kết tiến trình với một tài nguyên cục bộ có sẵn  
cùng loại. Chỉ khi một tài nguyên không tồn tại, ta mới cần sao chép hoặc di chuyển  
nguyên gốc tới đích mới, hoặc thiết lập một tham chiếu toàn cục.  
Di trú trong các hệ thống hỗn tạp  
Cho tới giờ, chúng ta ngầm định rằng các mã di trú có thể được thực thi dễ dàng ở máy  
đích. Giả định này được coi là hợp lệ khi xử lý với các hệ thống hỗn tạp. Một cách tổng  
quát, dĩ nhiên, các hệ phân tán được cấu trúc bởi một tập các platform hỗn tạp, đều có hệ  
điều hành riêng và kiến trúc máy tính riêng. Di trú trong mỗi hệ thống đó đòi hỏi mỗi  
platform được hỗ trợ, theo đó, mọt phân đoạn mã có thể được thực thi trên từng platform.  
Mặt khác, chúng ta cũng cần đảm bảo phân đoạn mã có thể tương ứng với mỗi platform.  
Các vấn đề đối với tính hỗn tạp trong nhiều khía cạnh giống với vấn đề về tính di động.  
và các giải pháp có phần tương đồng. Ví dụ, cuối những năm 1970, một giải pháp đơn  
giản để giảm thiểu các vấn đề trong ngôn ngữ Pascal giữa các máy khác nhau là tạo ra mã  
tức thì độc lập với máy tính cho các máy ảo trừu tượng (Barron 1981). Các máy ảo đó,  
  
không cần phải cài đặt trên các platform, nhưng nó vẫn cho phép các chương trình Pascal  
có thể chạy ở bất cứ đâu. Mặc dù ý tưởng đơn giản này được sử dụng rộng rãi trong nhiều  
năm, nó chưa bao giờ được coi là một giải pháp cơ bản đê giải quyết vấn đề di động cho  
các ngôn ngữ khác, nhất là C.  
Khoảng 25 năm sau, di trú mã trong các hệ thống hỗn tạp bị đe doạ bởi sự ra đời của các  
ngôn ngữ script và ngôn ngữ có tính di động cao như Java. Về thực chất, những giải pháp  
này cũng áp dụng cách giống như đã làm với di trú Pascal. Tất cả các giải pháp này đều  
có điểm chung là dựa vào một máy ảo để biên dịch trực tiếp mã nguồn (như trong trường  
hợp của các ngôn ngữ script), hoặc các biên dịch các mã được sinh ra bởi một trình biên  
dịch (như trong Java).  
Các bước phát triển gần đây đã bắt đầu làm giảm đáng kể tính phụ thuộc vào các ngôn  
ngữ lập trình. Cá biệt, các giải pháp đề xuất không chỉ di trú tiến trình, mà còn di trú toàn  
bộ môi trường tính toán. Ý tưởng cơ bản là phân chia toàn bộ môi trường và cung cấp các  
tiến trình trong cùng một phần tầm nhìn của chúng trong môi trường tính toán.  
Nếu việc phân chia được làm chính xác, nó dẫn tới khả năng có thể nhân đôi một phần từ  
hệ thống cơ sở và di trú tới một máy khác. Theo cách này, di trú thực sự cung cấp một  
form di động mạnh cho các tiến trình, chúng có thể được di chuyển ở mọi thời điểm thực  
thi, và tiếp tục khi di trú xong. Hơn nữa, nhiêu vấn đề khó hiểu liên quan đến tiến trình di  
trú khi chúng được gắn kết với các tài nguyên cục bộ có thể được giải quyết, các gắn kết  
này trong nhiều trường hợp đã được dự trữ sẵn. Các tài nguyên cục bộ thường là một  
phần của môi trường được di trú.  
Có một vài nguyên nhân cho việc di trú môi trường, nhưng dĩ nhiên điều quan trọng nhất  
là cho phép sự tiếp tục hoạt động khi một máy tính tắt. Ví dụ, trong một cluster server,  
quản trị hệ thống có thể quyết định cho tắt hoặc thay thế một máy tính, nhưng sẽ không  
phải ngừng tất cả các tiến trình đang chạy của nó. Thay vào đó, có thể tạm thời đóng  
băng môi trường, chuyển nó tới một máy khác và mở đóng băng trở lại. Đó thực sự là  
một cách mạnh mẽ để quản lý môi trường tính toán và các tiến trình của chúng.  
Ta hãy xét một ví dụ cụ thể về di trú máy ảo, như đã được đề cập bởi Clark(2005). Trong  
trường hợp này, tác giả tập trung vào di trú thời gian thực của một hệ điều hành ảo,  
thường là sẽ thuận lợi trong một cluster server khi liên kết chặt chẽ đạt tới thông qua một mạng cục bộ đơn và có chia sẻ. Trong các tình huống này, di trú sẽ có 2 vấn đề cơ bản: di  
trú toàn bộ ảnh bộ nhớ và di trú liên kết tới tài nguyên cục bộ.  
Với vấn đề đầu tiên, trong lý thuyết, ta có 3 cách đề quản lý di trú  
• Đẩy các trang bộ nhớ vào máy mới và gửi lại những trang đã được chỉnh sửa sau  
trong suốt tiến trình di trú.  
• Ngừng máy ảo hiện thời, di trú bộ nhớ, và khởi tạo máy ảo mới.  
• Để máy ảo mới yêu cầu mọt trang mới nếu cần, ta sẽ để các tiến trình khởi đầu  
trên máy ảo mới ngay lập tức.  
Lựa chọn thứ hai có thể dẫn tới thời gian chết quá lớn nếu các máy ảo di trú đang chạy  
dichjvuj trực tuyến, nghĩa là các dịch vụ đòi hỏi tính tiếp diễn liên tục. Mặt khác, một  
cách tiếp cận theo yêu cầu thuần túy theo phương pháp thứ ba có thể kéo dài chu kỳ di  
trú, nhưng sẽ dẫn tới hiệu năng thấp vì sẽ tốn kém nhiều thời gian trước khi làm việc với  
các tiến trình di trú được di chuyển tới máy mới.  
Như một lựa chọn khác, Clark đề xuất sử dụng cách tiền sao chép khi kết hợp lựa chọn  
đầu tiên, kèm theo một pha “dừng và sao chép”(stop-and-copy) như giới thiệu trong lựa  
chọn thứ hai. Kết quả của sự kết hợp này có thể dẫn tới thời gian chết không quá 0,2s.  
Liên quan đến tài nguyên cục bộ, vấn đề được đưa ra khi xử lý với một cluster server.  
Đầu tiên, bởi vì chỉ có một mạng đơn lẻ, điều cần làm chỉ là thông báo về sự gắn kết mới  
giữa mạng và địa chỉ MAC, để cho client có thể liên hệ các tiến trình di trú ở đúng card  
mạng.  
Hiệu quả tổng thể đó là, thay vì di trú tiến trình, chúng ta thực sự có thể thấy toàn bộ một  
hệ điều hành có thể được di chuyển giữa các máy tính khác nhau.  
3.3. Kết luận  
Ta đã thấy được vai trò quan trọng của di trú mã cho các hệ phân tán giữa các máy khác  
nhau. Hai nguyên nhân quan trọng để hỗ trợ di trú mã là tăng hiệu năng và tính linh hoạt.  
Khi truyền thông tốn kém, chúng ta có thể giảm chi phí bằng cách chuyển bớt tính toán từ  
server sang các client, và đề cho client thực hiện tính toán cục bộ các nhiều càng tốt. Tính  
linh hoạt tăng lên nếu một client có thể tự động tải về ềm cần thiết dể giao tiếp với server  
nhất định. Phần mềm được tải có thể định hướng sẵn tới server, mà không cần client phải  
cài đặt trước.  
Di trú mã đặt ra những vấn đề liên quan tới sử dụng tài nguyên cục bộ mà theo đó yêu  
cầu các tài nguyên khác được di trú, gắn kết mới với tài nguyên cục bộ ở máy đích được  
thiết lập, hoặc các tham chiếu toàn cục được sử dụng. Một vấn đề khác là di trú mã đòi  
hỏi tính hỗn tạp trong tài khoản. Các thí nghiệm hiện tại đưa ra phương án tốt nhất để xử  
lý tính hỗn tạp đó là sử dụng các máy ảo.

**4. Truyền thông**

Truyền thông đa tiến trình là trái tim của hệ phân tán. Hệ phân tán hiện đại chứa hàng  
nghìn, thậm chí hàng triệu tiến trình đồng thời hoạt động trong mạng, tiêu biểu như  
Internet. Trong phần này, ta sẽ xét tới vai trò của các mô hình truyền thông hiện đại trong  
việc đảm bảo tính trong suốt của hệ phân tán:  
• Gọi thủ tục từ xa RPC (Remote Procedure Call)  
• Triệu gọi đối tượng từ xa ROI(Remote Object Invocation)  
• Truyền thông điệp MOM (Message Oriented Middleware)  
• Truyền thông hướng dạng (Stream Oriented Middleware)  
4.1. Gọi thủ tục từ xa RPC  
Rất nhiều hệ phân tán dựa trên việc trao đổi thông điệp chi tiết giữa các tiến trình. Tuy  
nhiên các thủ tục truyền và nhận trong giao tiếp không hoàn toàn được che giấu, trong  
khi tính trong suốt trong truy cập là một trong những đặc tính quan trọng của hệ phân tán.  
Vấn đề này đã được đặt ra từ lâu nhưng chỉ cơ bản được giải quyết khi Birrell và Nelson  
(1984) đề ra một cách xử lý giao tiếp hoàn toàn khác.  
Hai người đã đề ra một số quy tắc cho phép chương trình gọi các hàm trên máy khác. Khi  
một tiến trình trên máy A gọi một hàm trên máy B, tiến trình trên máy A sẽ tạm thời bị  
treo và việc xử lý hàm được gọi được thực hiện trên máy B. Thông tin có thể truyền đến  
máy được gọi qua các tham số và trả kết quả về trong hàm kết quả. Không có bất kì thông  
điệp của tiến trình này biểu hiện ra bên ngoài đối với người lập trình. Đây chính là nội  
dung của phương pháp RPC.  
Mặc dù ý tưởng đưa ra có vẻ rất đơn giản và hiệu quả nhưng vẫn tồn tại một số vấn đề  
như : các hàm trên máy gọi và được gọi chạy trên các máy khác nhau và các máy này xử  
lý trên các không gian địa chỉ khác nhau , điều này sẽ tạo ra một số rắc rối,bên cạnh đó  
các tham số và kết quả phải được truyền giữa các máy , tuy nhiên nếu các máy này không  
15  
đồng bộ sẽ dẫn đến một số vấn đề rất phức tạp. Cuối cùng có thể một trong hai máy hoặc  
thậm chí cả hai máy đều hoạt động không đúng và mỗi lỗi lại gây ra cac vấn đề khác  
nhau. Tuy nhiên, hầu hết các vấn đề trên đều có thể xử lý được và RPC đang trở thành  
một công nghệ được sử dụng rộng rãi trên rất nhiều hệ phân tán.  
Hoạt động của RPC  
- Giải thích cách mà thủ tục có thể xử lý trên các máy khác nhau  
Các quy ước trong thủ tục gọi  
- Khi một thủ tục được thực hiện , các tham số của nó đươc đẩy vào trong ngăn xếp. Sau  
khi thủ tục được thực hiện xong , kết quả được lưu trong thanh ghi. Thủ được trả quyền  
điều khiển, các tham số trong ngăn xếp được xóa và ngăn xếp trở về trạng thái trước khi  
có lời gọi hàm.  
- Các cách truyền tham số:  
• Truyền bằng tham trị  
• Truyền bằng tham biến  
• Truyền bằng cách sao chép và phục hồi ( không phổ biến)  
Client stub và Server stub  
Cách thức hoạt động của RPC ( cách thức RPC hoạt động một cách trong suốt)  
- Client stub và server stub ở máy client và server là thành phần nhằm giảm nhẹ công  
việc cho client và server, làm cho hệ thống hoạt động một cách trong suốt hơn.  
- Trong mô hình client – server thì lời gọi thủ tục từ xa được thực hiện qua các bước sau:  
Một phiên làm việc theo giao thức RPC  
(1) Tiến trình muốn thực hiện thủ tục ở máy client sẽ gọi client stub.  
16  
(2) Client stub sẽ tạo một bản tin và có lời gọi đến hệ điều hành của client đó.  
(3) Hệ điều hành của máy client sẽ gửi bản tin đó tới hệ điều hành của máy  
server.  
(4) Hệ điều hành của server sẽ gửi bản tin tới server stub.  
(5) Server stub lấy các thông tin của gói tin và gọi server tương ứng.  
(6) Server thực hiện công việc được yêu cầu và trả kết quả về cho server stub.  
(7) Server stub đóng gói kết quả đó vào bản tin rồi gọi hệ điều hành của server đó.  
(8) Hệ điều hành của máy server này sẽ gửi bản tin kết quả đó hệ điều hành của  
máy client.  
(9) Hệ điều hành của máy client sẽ gửi bản tin cho client stub.  
(10) Client stub sẽ mở gói tin kết quả và trả về cho client.  
Truyền các tham số (Parameter Passing)  
Chức năng của Client stub là lấy các tham số, gói chúng lại trong một thông điệp rồi  
thông qua OS của Client gửi chúng cho Server.  
Truyền tham trị (Passing value parameter)  
• Parameter marshaling : Đưa các tham số vào một cách tuần tự.  
• Ngoài việc đóng gói các tham số của hàm , client stub còn đưa vào cả tên hoặc  
định danh của thủ tục được gọi vì server có thề hỗ trợ các hàm khác nhau, nó sẽ  
giúp server biết được hàm nào cần được gọi.  
• Thông điệp gửi đến server, server xác định thủ tục được gọi theo trường đầu  
tiên của thông điệp.Khi server hoàn thành công việc kết quả được tra về cho  
client.  
• Nếu client và server đồng bộ và các tham số và kết quả là các đại lượng vô  
hướng hoặc Boolean thì hệ thống sẽ làm việc hoàn toàn đúng đắn.Tuy nhiên  
trong các hệ thống lớn sẽ có rất nhiều các loại máy khác nhau, mỗi dòng máy có  
một cách biểu diễn các số, các ký tự và các kiểu dữ liệu khác nhau, điều này dẫn  
đến việc hiểu nhầm giữa client và server.  
• Nếu không có các thông tin thêm thì gần như là không thể truyền tham số từ  
một Client này tới một Server khác.  
Truyền tham trị (Passing value parameter )  
17  
Đây là một vấn đề khó: cách để truyền con trỏ và tham chiếu, vì chúng chỉ có ý nghĩa  
trong không gian địa chỉ mà tiến trình sử dụng( trong đó có chứa địa chỉ mà chúng trỏ  
đến).  
Có hai cách giải quyết  
- Cách thứ nhất:Cấm việc sử dụng con trỏ và tham chiếu. Tuy nhiên cách này  
không được ưa chuộng.  
- Cách thứ hai: nếu ta biết kích thước của mảng dữ liệu. Ta sẽ copy luôn mảng dữ  
liệu đó và gửi cho server .Khi đó server sẽ sử dụng được con trỏ và tham chiếu  
đến chính mảng dữ liệu này. Sau khi server hoàn thành công việc mảng này sẽ  
được gửi lại cho client stub sau đó copy lại cho client.Trong trường hợp này  
truyền tham chiếu được thay thế bằng cách truyền theo kiểu sao chép và phục hồi.  
- Mặc dù chúng ta đã xử lý được con trỏ với cấu trúc mảng đơn giản , nhưng chúng  
ta vẫn chưa xử lý được phần lớn các trường hợp con trỏ với cấu trúc dữ liệu tùy  
biến ví dụ như bản đồ  
Chi tiết hóa các tham số và sinh Stub (Parameter Specification and Stub Generation)  
Giải pháp cho các vấn đề trên  
♣ Khi thực hiện một RPC thì cả Client lẫn Server phải thống nhất với nhau về định  
dạng của thông điệp sẽ được trao đổi giữa chúng.Chúng sẽ phải theo một giao  
thức chung, nếu không PRC sẽ làm việc không đúng đắn  
♣ Định nghĩa cấu trúc dữ liệu là một khía cạnh của giao thức RPC tuy nhiên chưa  
đủ. Điều ta cần đến là sự thỏa thuận giữa Client và Server về mô tả các cấu trúc  
dữ liệu đơn giản Integer, Characters, Boolean…( ví dụ, giao thức sẽ yêu cầu biểu  
diễn các kí tự dưới dạng 16 bit Unicode và float được định dạng theo chuẩn  
IEEE754, và dữ liệu được lưu trữ theo kiểu đầu nhỏ …) . Với những thông tin  
thêm này, thông điệp được tạo ra rất rõ ràng. Thêm một điều nữa, cả client và  
server sẽ phải thỏa thuận về cách truyền thông điệp ( ví dụ sử dụng truyền thông  
hướng kết nỗi như TCP/IP).  
♣ Trong thực hành ta thấy rằng là sử dụng ngôn ngữ định nghĩa giao diện IDL  
(Interface Definition Language) gần như đơn giản hóa các ứng dụng Client-  
Server dựa trên RPC.  
18  
RPC dị bộ (Asynchronous RPC)  
• Trong hoạt động của một RPC thì khi Client gửi thông điệp (gọi thủ tục) nó sẽ  
dừng lại chờ kết quả trả về từ Server, tuy nhiên việc này gây ra sự lãng phí phía  
Client, nhất là khi có nhiều việc cần Client phải thực hiện.  
• Trong gọi thủ tục từ xa không đồng bộ thì Client sẽ tiếp tục làm việc ngay mà  
không phải chờ cho tới khi nó nhận được phản hồi từ phía Server. Phản hồi này  
được gửi tới Client ngay khi Server nhận được thông điệp từ Client, và sau đó  
Server tiếp tục xử lý cho ra kết quả rồi gửi trở lại Client.  
RPC đồng bộ trễ (Deferred synchronuos RPC)  
Thực hiện hai lời gọi, một từ client và một từ server. Client gửi tới server lời gọi  
thủ tục và chờ bản tin chấp nhận từ server. Phía server sẽ gửi bản tin chấp nhận về cho  
client thông báo đã nhận được yêu cầu và bắt đầu thực hiện yêu cầu RPC đó. Lúc này  
client sẽ tiếp tục thực hiện công việc của mình. Khi thực hiện thủ tục xong, server sẽ thực  
hiện lời gọi tới client báo nhận lấy kết quả. Client thực hiện ngắt, nhận kết quả và gửi lại  
cho server bản tin thông báo đã nhận kết quả thành công.  
19  
RPC đồng bộ trễ  
RPC đơn tuyến (one- way RPC)  
Sau khi thực hiện lời gọi thủ tục từ xa tới server, client không chờ đợi thông báo  
nhận yêu cầu thành công từ server mà tiếp tục thực hiện ngay các công việc khác của  
mình. Đó là RPC đơn tuyến. Tuy nhiên điều này không đảm bảo độ tin cậy do client  
không biết liệu yêu cầu có được thực hiện hay không. Trong trường hợp RPC đồng bộ bị  
hoãn lại, client có thể hỏi server xem kết quả có được thực hiện hay không mà không đợi  
server thông báo lại cho client.  
5. Định danh  
Tên giữ một vai trò rất quan trọng trong tất cả các hệ thống máy tính. Chúng thường được  
sử dụng để chia sẻ tài nguyên, nhận biết các thực thể duy nhất, tương ứng với các vị trị và  
hơn nữa. Một vấn đề quan trọng của việc định danh đó là tên có thể được phân giải tới  
thực thể tương ứng. Ngoài ra, việc định danh cũng có vai trò quan trọng trong việc đảm  
bảo tính trong suốt về vị trí ytong hệ phân tán. Giải pháp định danh cho phép một tiến  
trình truy cập tới thực thể đã được đặt tên. Để phân giải tên, cần phải bổ sung hệ thống  
định danh. Sự khác nhau giữa việc đặt tên trong hệ phân tán và hệ không phân tán nằm ở  
cách hệ thống đặt tên được thực hiện.  
Một số kiểu định danh :  
Tên (Name)  
Địa chỉ (Address) Định danh (ID)  
Một xâu bit/char để  
tham chiếu đến 1 thực  
thể trong hệ phân tán  
Một tên đặc biệt có các tính chất :  
Một ID tham chiếu tới nhiều nhất  
một thực thể  
Mỗi thực thể được tham chiếu bởi  
nhiều nhất một ID  
Một ID luôn tham chiếu tới cung  
một thực thể  
Một xâu bit/char để  
tham chiếu đến 1 thực  
thể trong hệ phân tán  
20  
• Định danh phẳng (Flat Naming): ID là chuỗi bit ngẫu nhiên, phi cấu trúc. Một  
tính chất quan trọng là nó không chứa bất cứ thông tin nào về vị trí điểm truy  
cập của thực thể liên kết.  
• Định danh có cấu trúc (Structured Naming): Định danh phẳng thích hợp với  
máy nhưng nó gây khó khăn cho người sử dụng. Do đó ra đời một hệ thống  
định danh hỗ trợ các tên có cấu trúc, phù hợp với con người, phục vụ không chỉ  
hệ thống tên file mà cả hệ thống tên host trên internet.  
• Định danh theo thuộc tính là cách mô tả một thực thể theo một cặp (thuộc tính,  
giá trị). Theo cách định danh này, một thực thể có một tập hợp liên kết các thuộc  
tính, mỗi thuộc tính lại mô tả một vài điều về thực thể này. Bằng việc chỉ rõ ra  
giá trị mà thuộc tính có thể có, người sử dụng ràng buộc bản chất của các thực thể  
mà anh ta quan tâm. Công việc của hệ thống định danh là trả về một hay nhiều  
thực thể hợp với những mô tả đã được người sử dụng cung cấp.  
6. Đồng bộ hoá  
Trong hệ phân tán, vai trò của vấn để đảm bảo tính ổn định và thống nhất hoạt động.  
chúng ta sẽ tìm hiểu bằng cách nào các tiến trình đồng bộ hóa được với nhau. Ví dụ, thay  
vì nhiều tiến trình đồng thời truy nhập vào một tài nguyên chia sẻ thì chúng cấp quyền  
truy nhập tạm thời cho nhau. Một ví dụ khác, nhiều tiến trình đôi khi cần trả lời cho 1 sự  
21  
kiện nào đó, nói cách khác, cần xác định thông điệp m1 của tiến trình P được gửi trước  
hay sau thông điệp m2 cùa tiến trình Q.  
Đồng bộ hóa trong các hệ thống phân tán thường khó hơn rất nhiều so với đồng bộ hóa  
trong các hệ đơn hoặc đa xử lý.  
Ta sẽ hướng tới đồng hộ hóa dựa trên thời gian hoạt động, tức là thời gian có tính tương  
quan giữa các tiến trình hơn là thời gian tuyệt đối.  
Trong nhiều trường hợp, đồng bộ hóa có thể được giải quyết bằng cách một nhóm các  
tiến trình có thể sử dụng 1 tiến trình được thực thi bằng cách lấy trung bình một vài thuật  
toán lựa chọn.  
7. Tính nhất quán và sử dụng bản sao  
Dữ liệu nói chung trong hệ phân tán được nhân lên thành nhiều bản để tăng thêm tính tin  
cậy và tăng hiệu năng. Vấn đề chính của các bản sao là sự nhất quán giữa các bản sao khi  
một hoặc 1 số bản sao bị thay đổi. Ta cần quan tâm tới 2 vấn đề : quản lý bản sao và giữ  
các bản sao được nhất quán khi xây dựng hệ phân tán.  
• Để tăng tín tin cậy cho hệ thống : Trong quá trình đọc hoặc ghi dữ liệu , nếu một  
bản sao bị hỏng hoặc vì lý do nào đấy nó không sẵn sàng để dùng thì ta có thể sử  
dụng một bản khác và tránh việc sử dụng dữ liệu sai lạc, không chính xác.  
• Để tăng hiệu năng của hệ thống : Tăng quy mô của hệ thống cả về số lượng lẫn  
phạm vi địa lý, như nếu số lượng máy tăng thì tải trên mỗi máy trong hệ phân tán  
cũng không tăng lên nhiều hơn. Nếu phạm vi địa lý tăng ta có thể sử dụng các bản  
nhân bản gần khu vực địa lý đó .  
Một cái giá phải trả cho việc nhân bản dữ liệu đấy là phải chắc chắn rằng khi một bản sao  
được cập nhật thì tất cả những bản còn lại cũng phải được cập nhật theo để đảm bảo tính  
nhất quán của dữ liệu. Điều này ảnh hưởng rất lớn đến hiệu năng của hệ thống.  
Ví dụ : khi sử dụng các chương trình xem web => dữ liệu được ghi vào bộ nhớ máy tính (  
bản sao ) , nếu trang web nội dung thay đổi . người dùng vẫn chỉ xem nội dung trên máy  
=> cũ . Vấn đề đồng bộ dữ liệu trên máy khách và trên máy chủ  
Có 2 hướng tiếp cận cho việc nhân bản đối tượng.  
¬ Ứng dụng tự chịu trách nhiệm nhân bản. Do đó ứng dụng cần giải quyết được vấn  
đề nhất quán.  
22  
¬ Hệ thống trung gian đảm trách việc nhân bản. Vì thế vấn đề nhất quán do  
middleware đảm trách. Hướng này đơn giản hóa cho việc phát triển ứng dụng  
nhưng lại khiến các giải pháp đặc tả đối tượng trở nên phức tạp hơn.  
8.Tính chịu lỗi (Fault Tolerance)  
Một đặc tính của hệ phân tán khác biệt với các hệ thống máy đơn lẻ là khái niệm lỗi bộ  
phận, xảy ra khi một thành phần của hệ thống gặp sự cố. Lỗi này có thể ảnh hưởng đến  
hoạt động của các thành phần khác, trong khi một số thành phần không chịu bất cứ ảnh  
hưởng nào. Ngược lại, sự cố trong các hệ không phân tán thường làm cả hệ thống phải  
ngừng hoạt động.  
Mục đích quan trọng của hệ phân tán là xây dựng theo cách có thể tự động khôi phục lỗi  
mà không ảnh hưởng đến hiệu năng hoạt động. Khi gặp sự cố, hệ thống có thể tiếp tục  
hoạt động ở mức chấp nhận được trong khi sửa lỗi được tiến hành ngay, có nghĩa là có  
thể chịu lỗi và tiếp tục thực thi thêm một số sự kiện dù có lỗi.  
8.1. Một số định nghĩa  
Một hệ thống có khả năng chịu lỗi là một hệ thống đảm bảo được những yêu cầu sau:  
• Tính sẵn sàng  
• Tính tin cậy  
• Tính an toàn  
• Khả năng duy trì hoạt động (maintainability).  
Tính sẵn sàng nghĩa là hệ thống có thể sử dụng ngay lập tức. Nói chung điều này thể hiện  
khả năng hệ thống hoạt động chính xác trong bất kỳ thời điểm nào và sẵn sàng thực hiện  
chức năng trong khả năng.  
Tính tin cậy nghĩa là hệ thống có thể chạy liên tục mà không phát sinh lỗi. Khác với tính  
sẵn sàng, một hệ thống tin cậy cao là một hệ thống có thể hoạt động liên tục mà không có  
bất kỳ một sự gián đoạn nào trong một khoảng thời gian dài. Đây là một sự khác biệt khó  
nhận ra nhưng rất quan trọng khi đem so sánh với tính sẵn sàng. Nếu một hệ thống ở  
trạng thái down 1 milisecond mỗi giờ, tính sẵn sàng của nó đạt đến 99,9999 % nhưng vẫn  
23  
là một hệ thống không tin cậy. Ngược lại một hệ thống không bao giờ đổ vỡ nhưng luôn  
luôn ở trạng thái ngừng trong 2 tuần của một tháng nhất định là một hệ thống tin cậy cao  
nhưng lại chỉ đạt được 96% sẵn sàng. Tính tin cậy và tính sẵn sàng không giống nhau.  
Tính an toàn thể hiện ở chỗ khi hệ thống tạm thời bị lỗi, không có thiệt hại nghiêm trọng  
nào xảy ra. Chẳng hạn nhiều hệ thống điều khiển tiến trình như hệ thống dùng để điều  
khiển nhà máy hạt nhân hay đưa con người vào vũ trụ yêu cầu độ an toàn cao. Nếu hệ  
thống điều khiển chỉ bị lỗi trong một khoảng thời gian rất ngắn, hậu quả có thể rất thảm  
khốc. Nhiều ví dụ trong quá khứ đã chứng tỏ rất khó để xây dựng một hệ thống an toàn.  
Cuối cùng, tính duy trì thể hiện ở chỗ một hệ thống lỗi có thể được sửa một cách dễ dàng.  
Một hệ thống có tính duy trì cao sẽ có tính sẵn sàng cao, đặc biệt là nế lỗi có thể được  
phát hiện và sửa chữa một cách tự động. Tuy nhiên như chúng ta sẽ thấy sau trong  
chương này, việc tự động phục hồi lỗi là rất khó.  
Thông thường một hệ thống đáng tin cậy còn đòi hỏi phải cung cấp được độ an toàn an  
ninh cao, đặc biệt khi nó đi đến vấn đề như tính toàn vẹn.  
Một hệ thống bị coi là lỗi khi nó không thể thực hiện được những chức năng thông  
thường của nó. Cụ thể nếu một hệ phân tán được thiết kế để cung cấp một số những dịch  
vụ, hệ thống gặp lỗi khi nó không thể cung cấp được một trong những dịch vụ đó.  
Rõ ràng việc tìm ra nguyên nhân gây lỗi là rất quan trọng. Chẳng hạn một môi trường  
truyền không tốt có thể dễ dàng ảnh hưởng đến. Trong trường hợp này, xóa bỏ lỗi là khá  
dễ dàng. Tuy nhiên lỗi do truyền có thể bị gây ra bởi điều kiện thời tiết xấu (ví dụ trong  
mạng wireless). Thay đổi thời tiết để ngăn chặn lỗi là một giải pháp không khả thi.  
Việc xây dựng một hệ thống có thể tin cậy được liên quan chặt chẽ đến việc xử lý lỗi.  
Với chúng ta, điều quan trọng nhất là tính chịu lỗi, nghĩa là hệ thống có thể cung cấp các  
dịch vụ trong khi vẫn đang gặp lỗi. Nói cách khác, hệ thống có thể chịu lỗi và tiếp tục  
hoạt động một cách bình thường.  
Lỗi thường được chia thành 3 loại: nhất thời, liên tiếp hoặc lâu dài.  
• Lỗi nhất thời chỉ xuất hiện một lần rồi biến mất. Nếu quá trình hoạt động lặp lại,  
lỗi không xuất hiện nữa.  
• Lỗi liên tiếp là tình trạng hoạt động không ổn định, lỗi lặp đi lặp lại nhiều lần. Lỗi  
liên tiếp là nguyên nhân của những hậu quả nghiêm trọng vì khó tìm được nguyên  
nhân.  
24  
• Lỗi lâu dài là lỗi chỉ được khắc phục khi thành phần gây lỗi được thay thế, ví dụ  
như cháy nổ chip, lỗi phần mềm, lỗi ổ đĩa  
8.2. Các mô hình lỗi  
Một hệ thống lỗi là khi nó không cung cấp đầy đủ các dịch vụ như thiết kế. Nếu coi một  
hệ phân tán là một tập các server giao tiếp với nhau và với các client thì không cung cấp  
đầy đủ các dịch vụ nghĩa là các server, các kênh truyền thông, hoặc cả 2 không thực hiện  
đúng nhiệm vụ của nó. Tuy nhiên một server hoạt động sai chức năng chưa chắc đã là  
nguyên nhân gây ra lỗi. Nếu một server phụ thuộc vào các server khác để cung cấp đầy  
đủ các dịch vụ của nó, nguyên nhân của lỗi có thể cần phải được tìm kiếm ở những nơi  
khác nữa ngoài server đó, mặc dù server đó bị lỗi.  
Mối quan hệ phụ thuộc đó xuất hiện rất thường xuyên trong hệ phân tán. Một đĩa cứng bị  
lỗi có thể ảnh hưởng đến một file server được thiết kế để cung cấp hệ thống file có tính  
sẵn sàng cao. Nếu một file server như vậy là một phần của một hệ cơ sở dữ liệu phân tán,  
sự hoạt động chính xác của hệ toàn bộ hệ cơ sở dữ liệu có thể bị đe dọa và chỉ một phần  
dữ liệu là có thể truy cập được.  
Để hiểu rõ hơn thực tế một lỗi là nghiêm trọng đến mức nào, người ta đã đưa ra một vài  
cách phân loại như sau:  
• Lỗi sụp đổ (Crash failure) xảy ra khi một server ngừng hoạt động trước dự kiến,  
nhưng vẫn làm việc chính xác cho đến khi nó dừng. Một ví dụ điển hình của  
25