Tạp chí Tin học và Điều khiển học, T.21, S.4 (2005), 323–336

MÔ HÌNH PHẦN MỀM CHỊU LỖI BK-FTS: THỬ NGHIỆM, SO SÁNH VÀ ĐÁNH GIÁ

PHẠM BÁ QUANG, ĐÀO NGỌC KIÊN, HUỲNH QUYẾT THẮNG

Khoa CNTT, Trường Đại học Bách khoa Hà Nội

Abstract. This paper gives an overview of recent methods and approach to analyze and develop fault tolerant software. We present the architecture of fault tolerant software BK-FTS and results of using this architecture to build a flying management system. The architecture, components and mechanism operations were go into details. The experimental results realized on the proposed BK-FTS architecture by various methods of replica management also presented.

Tóm tắt. Bài báo đề cập đến tư tưởng, phương pháp và các kỹ thuật áp dụng trong xây dựng phần mềm chịu lỗi và giới thiệu một số mô hình phần mềm chịu lỗi. Trên cơ sở nghiên cứu các mô hình phần mềm chịu lỗi, phân tích các tồn tại trong các mô hình này, chúng tôi đề xuất mô hình phần mềm chịu lỗi BK-FTS. Kiến trúc, các thành phần, cơ chế hoạt động của mô hình BK-FTS được trình bày chi tiết. Chúng tôi tiến hành thử nghiệm trên phần mềm mô phỏng kiểm soát không lưu, xây dựng theo mô hình phần mềm chịu lỗi BK-FTS. Các kết quả thử nghiệm, so sánh và đánh giá mô hình BK-FTS với các mô hình khác cũng được trình bày trong bài báo.

ĐẶT VẤN ĐỀ

Trong hoạt động của các hệ thống máy tính, độ an toàn và tính tin cậy là yêu cầu cơ bản và rất quan trọng. Bên cạnh phần cứng, phần mềm chiếm vị trí quan trọng đảm bảo cho hệ thống hoạt động một cách bình thường. Tăng cường độ tin cậy của phần mềm là một thách thức rất lớn so với phần cứng. Khác với quá trình sửa lỗi phần cứng - thay thế phần cứng bị hỏng bằng một phần cứng khác, quá trình thay đổi hoặc sửa chữa lỗi phần mềm có thể sinh ra các lỗi phần mềm mới khác và đặc biệt là không thể đảm bảo rằng phần mềm đã hết lỗi. Trong kỹ thuật phát triển phần mềm, người ta nghĩ đến việc nghiên cứu các giải pháp kỹ thuật phục vụ cho việc xây dựng phần mềm có khả năng chịu lỗi [1].

Bài báo này trình bày về tư tưởng, một số phương pháp xây dựng phần mềm chịu lỗi, mô hình phần mềm chịu lỗi, kết quả nghiên cứu trong hoàn thiện một mô hình phần mềm chịu lỗi BK-FTS: phân tích mô hình, các thử nghiệm và kết quả đánh giá, các định hướng nghiên cứu triển khai tiếp theo.

Mục 1 giới thiệu cơ sở lý thuyết xây dựng phần mềm chịu lỗi và một số kết quả nghiên cứu xây dưng mô hình phần mềm chịu lỗi hiện nay. Mục 2 trình bày mô hình phần mềm chịu lỗi BK-FTS đề xuất. Chúng tôi trình bày chi tiết kiến trúc, cơ chế hoạt động của từng thành phần trong mô hình. Mục 3 là các kết quả thử nghiệm, đánh giá tính đúng đắn, hiệu quả của mô hình đề xuất và một số vấn đề cần quan tâm chú ý. Cuối cùng, phần kết luận và định hướng nghiên cứu phát triển tiếp theo được trình bày trong Mục 4.

1. LÝ THUYẾT XÂY DỰNG PHẦN MỀM CHỊU LỖI VÀ MỘT SỐ MÔ HÌNH PHẦN MỀM CHỊU LỖI

1.1. Xây dựng phần mềm chịu lỗi

Các biện pháp được sử dụng để nâng cao độ tin cậy của phần mềm được chia thành hai nhóm chính [2]: Nhóm thứ nhất là các biện pháp được thực hiện trong quá trình xây dựng phần mềm, đó là biện pháp tránh lỗi (avoidance fault) và biện pháp xây dựng phần mềm có khả năng chịu lỗi (fault tolerance). Khả năng chịu lỗi của phần mềm thể hiện khả năng phần mềm vẫn tiếp tục hoạt động bình thường mặc dù có lỗi xảy ra. Nhóm thứ hai là các biện pháp kiểm soát phần mềm sau khi phần mềm được phát triển, bao gồm biện pháp loại bỏ lỗi (fault removal) và biện pháp dự đoán lỗi (fault forecasting). Ý tưởng chính của việc xây dựng bất kỳ một hệ thống chịu lỗi nào là có sẵn một số tài nguyên để làm dự phòng cho hệ thống, có thể là dự phòng về phần cứng và dự phòng về phần mềm. Trong bài báo này chúng tôi đề cập đến dự phòng về phần mềm. Theo [2] có ba nhóm phương pháp xây dựng phần mềm chịu lỗi:

• Phương pháp sử dụng một phiên bản phần mềm: bao gồm các kỹ thuật cổ điển, thường được áp dụng trong khi phát triển phần mềm như kỹ thuật bắt lỗi, quản lý ngoại lệ...

• Phương pháp sử dụng nhiều phiên bản phần mềm (đa thiết kế - design diversity): nhiều phiên bản phần mềm được thiết kế theo những cách khác nhau cùng thực hiện một công việc trong bài toán [1]. Dữ liệu đầu vào được cung cấp cho nhiều biến thể cùng hoạt động, các kết quả đầu ra được tổng hợp lại và đưa qua một bộ quyết định để xác định biến thể nào cho kết quả đúng rồi sau đó chuyển cho phần xử lý tiếp theo [1]. Các biến thể phần mềm phải được thiết kế và phát triển hoàn toàn độc lập và khác nhau, không chỉ đơn thuần là sao chép một biến thể thành nhiều biến thể cùng một cách thiết kế và phát triển.

• Phương pháp sử dụng nhiều dữ liệu (đa dữ liệu - data diversity): Phương pháp đa dữ liệu thực hiện biến đổi dữ liệu đầu vào của hệ thống thành nhiều đầu vào khác nhau. Thông qua phương pháp này người thiết kế mong muốn từ một đầu vào nằm trong tập các dữ liệu có khả năng phát sinh lỗi sẽ biến đổi thành một đầu vào khác nằm ngoài tập dữ liệu có khả năng phát sinh lỗi [1].

1.2. Các mô hình phần mềm chịu lỗi

Các mô hình phần mềm chịu lỗi hiện nay đang được nhiều người quan tâm và xây dựng. Trong [8] đề cập đến mô hình phần mềm chịu lỗi AFT-CCM (Adaptive Fault-Tolerance in the CORBA Component Model), một mô hình phục vụ cho việc xây dựng các ứng dụng dựa nền thành phần với các yêu cầu về chất lượng dịch vụ (QoS) liên quan đến tính chịu lỗi. AFT-CCM được xây dựng dựa trên mô hình thành phần CORBA - CCM (CORBA Component Model) - mô hình mở rộng của mô hình đối tượng CORBA Object Model do tổ chức OMG (Object Management Group www.omg.com) đưa ra. Mô hình AFT-CCM định nghĩa các đặc điểm như số lượng nhân bản hay kỹ thuật nhân bản nhằm đáp ứng các yêu cầu chịu lỗi cụ thể với từng người dùng. Các thành phần của AFT-CCM bao gồm: bộ quản lý lỗi (adaptive fault-tolerance manager - AFT manager) định nghĩa cấu hình hiện tại dựa trên cơ sở các yêu cầu chất lượng dịch vụ của ứng dụng; bộ điều phối nhân bản (replication coordinator) có nhiệm vụ thực thi các giao thức nhằm đảm bảo tính nhất quán; tác tử kiểm soát lỗi (fault detection agents - FD agent) có nhiệm vụ phát hiện các lỗi phần mềm xuất

hiện tại các môđun.

FT-CORBA là một mô hình phần mềm chịu lỗi cho các hệ thống phần mềm phân tán có tính sẵn dùng cao, có khả năng phục hồi sau khi lỗi [2,10]. Mô hình FT-CORBA bao gồm các thành phần chính: bộ quản lý nhân bản, bộ phát hiện lỗi, bộ lưu thông tin và khắc phục lỗi. Bộ quản lý nhân bản là một thành phần quan trọng chịu trách nhiệm quản lý nhân bản các đối tượng khi có phát sinh lỗi [10]. Một bộ quản lý nhân bản bao gồm các thành phần sau: PropertyManager - cho phép thiết lập các thuộc tính cho một nhóm đối tượng, các thuộc tính điển hình như kiểu nhân bản, kiểu quan hệ giữa các đối tượng, số lượng đối tượng nhân bản nhỏ nhất hay số lượng đối tượng khi khởi tạo; GenericFactory - được bộ quản lý nhân bản dùng để tạo ra các nhóm đối tượng hay để tạo ra các thành viên riêng lẻ của một nhóm đối tượng; ObjectGroupManager - được dùng bởi các ứng dụng để tạo mới, thêm vào hay xóa đi các thành viên trong một nhóm đối tượng. Bộ phát hiện lỗi gồm những bộ được cung cấp bởi hạ tầng cơ sở để theo dõi các đối tượng và các bộ cung cấp bởi ứng dụng, được xây dựng dựa trên cơ chế timeout và sử dụng cơ chế theo dõi Pull [10]. Cơ chế lưu thông tin và khắc phục lỗi trong khi đang hoạt động bình thường sẽ ghi lại các thông tin về trạng thái và hành động của bản sao chính. Khi bị lỗi thì cơ chế khắc phục lỗi lấy các thông tin này và dùng chúng để khôi phục lại hệ thống bằng cách thiết lập thông tin đó cho bản sao dự phòng nhờ đó mà Bản sao dự phòng có thể tiếp tục các dịch vụ bản sao chính trước đó đã cung cấp.

Mô hình CORBA Trading service do René Meier đề xuất [9]. Thành phần chính của mô hình này là CORBA Object Trading Service được dùng như một cơ chế thông báo và quản lý các dịch vụ hỗ trợ chịu lỗi [9]. Cơ chế này giúp các client có thể phát hiện được kết nối hỏng với một đối tượng dịch vụ, tìm ra các đối tượng dịch vụ dự phòng tương tự và thực hiện kết nối lại, giúp cải thiện sự ổn định cho toàn bộ hệ thống. Trong mô hình này Trading Service cho phép client phát hiện động các service object dựa trên loại dịch vụ mà chúng cung cấp. Nó giống như những trang vàng cho các service object. Server thông báo các service object của nó với một Trading Service và các client sẽ sử dụng Trading Service để tìm ra dịch vụ mà nó cần [9].

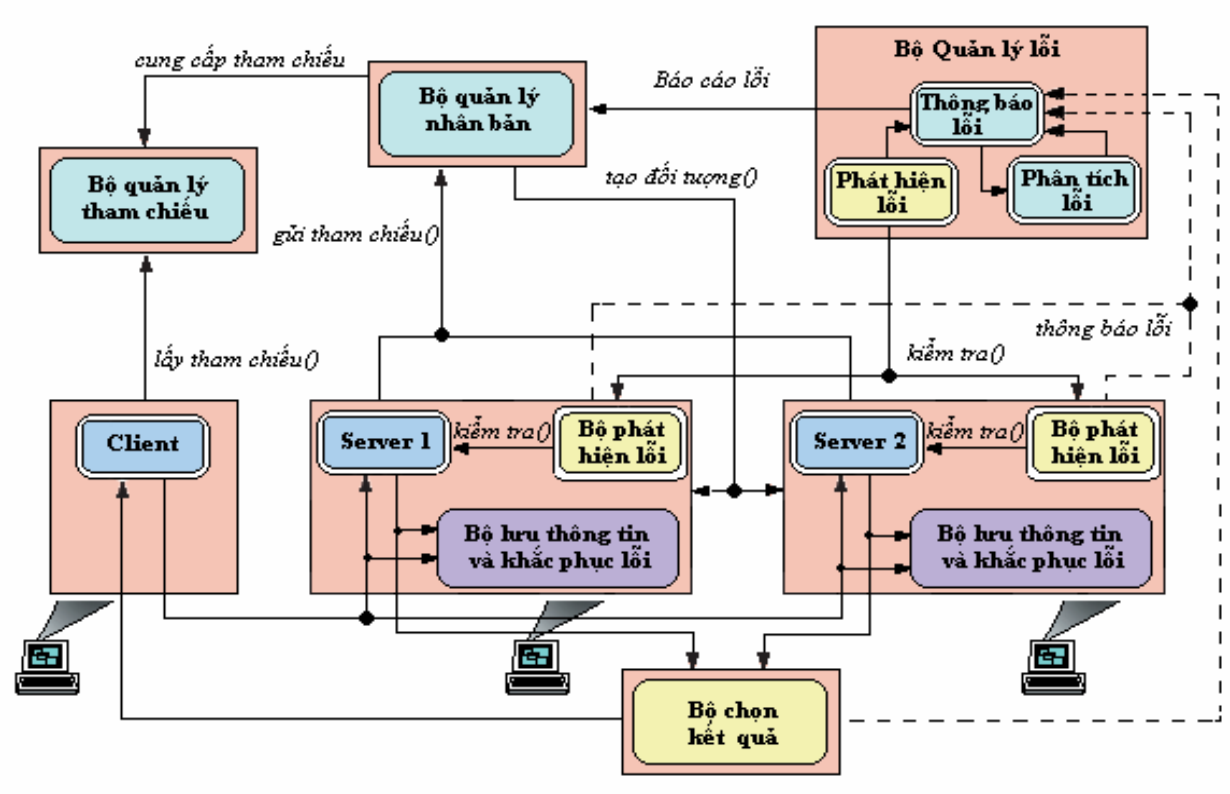
Nhìn chung các mô hình đều giống nhau về tư tưởng dư thừa phần mềm. Các mô hình đều cung cấp và hỗ trợ tính chịu lỗi cho các ứng dụng có yêu cầu độ tin cậy cao, dựa trên tư tưởng dư thừa thực thể, khả năng tự động phát hiện lỗi và tự động khắc phục lỗi. Trong các mô hình này, một đối tượng client có thể yêu cầu một tác vụ của đối tượng server mà không quan tâm đến vị trí vật lý của chúng. Các mô hình đã giới thiệu đều phù hợp với nhiều bài toán ứng dụng phân tán khác nhau. Mô hình Trading Service thì được thiết kế phù hợp với bài toán ngân hàng hơn cả tuy nhiên cũng có thể áp dụng với các bài toán tương tự. Mô hình AFT-CCM không đề cập đến khả năng chịu lỗi dữ liệu, không tổ chức phát hiện lỗi phân cấp, mô hình FT-CORBA không cho phép chịu lỗi dữ liệu, mô hình dựa Trading Service không đề cập đến khả năng phát hiện lỗi. Sự xuất hiện của bộ phân tích lỗi là khá quan trọng bởi trong một số hệ thống thì một số lỗi mà bộ phát hiện lỗi phát hiện ra cần được xử lý theo đặc thù riêng của hệ thống đó. Cách thiết kế và giải thuật thực thi bộ phân tích lỗi là tùy thuộc vào những đặc thù đó. Mô hình FT-CORBA có cung cấp cơ chế này, mô hình AFT-CCM và Trading Service lại không cho phép áp dụng các cơ chế này [2,10].

2. MÔ HÌNH PHẦN MỀM CHỊU LỖI BK-FTS

Dựa trên các mô hình phần mềm chịu lỗi và kế thừa các ý tưởng của mô hình FT-CORBA chúng tôi đề xuất xây dựng một mô hình phần mềm chịu lỗi nhằm mục đích phát triển cho các hệ thống yêu cầu độ tin cậy cao gọi tên là BK-FTS.

2.1. Mô hình BK-FTS

Các thành phần của mô hình BK-FTS được thể hiện trong Hình 1. Mô hình bao gồm các thành phần ứng dụng và các thành phần bổ sung: bộ quản lý nhân bản, bộ quản lý lỗi, bộ quản lý tham chiếu, bộ phát hiện lỗi nhằm cung cấp khả năng chịu lỗi cho hệ thống. Các thành phần ứng dụng là các biến thể phần mềm được phát triển khác nhau phục vụ cho mục đích bài toán. Các thành phần bổ sung sẽ phụ trách việc quản lý nhân bản, quản lý lỗi và khắc phục lỗi cho hệ thống. Mô hình được xây dựng dựa trên nguyên lý dư thừa thực thể (entity redundancy), phương pháp đa phiên bản (multiple version), khả năng phát hiện lỗi và khắc phục lỗi (fault detection, fault recovery). Dư thừa thực thể đạt được nhờ việc nhân bản các đối tượng (object). Một đối tượng ở đây là một biến thể phần mềm. Các đối tượng được kết hợp tạo thành một nhóm đối tượng (object group). Các thành viên trong một nhóm đối tượng (gọi là replica - bản sao) có giao diện giống nhau tuy nhiên phương pháp thiết kế là khác nhau. Chi tiết về cấu trúc các thành phần bộ quản lý nhân bản, bộ quản lý lỗi, bộ quản lý tham chiếu, bộ phát hiện lỗi và hoạt động sẽ được trình bày trong các mục tiếp theo.

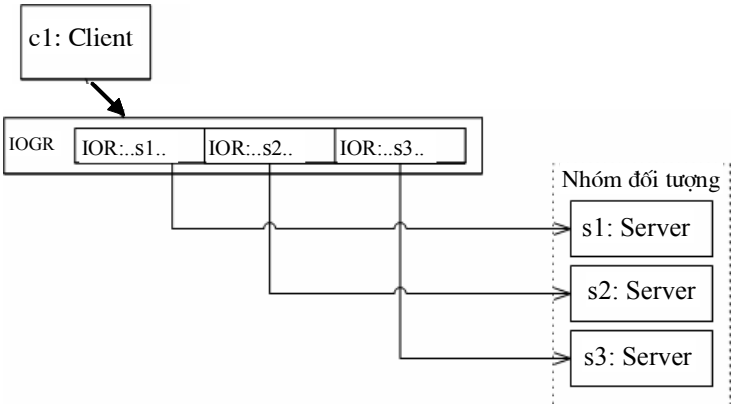


Hình 1. Mô hình BK-FTS

2.2. Cơ chế hoạt động của mô hình BK-FTS

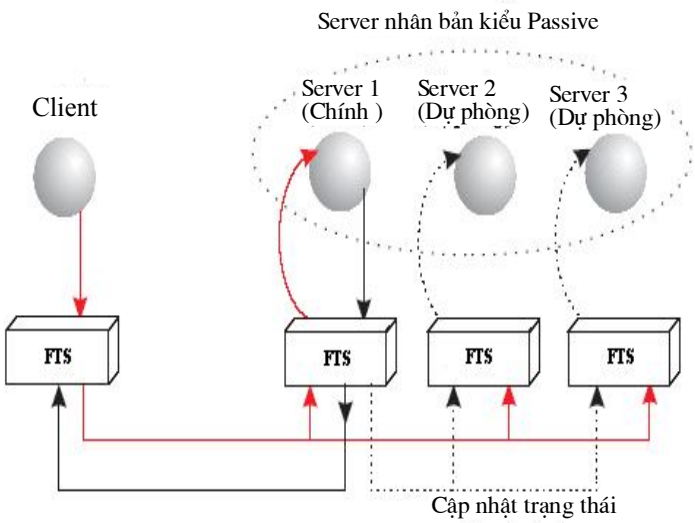
2.2.1. Các kiểu nhân bản

Mô hình BK-FTS được xây dựng áp dụng tư tưởng "dự phòng" và phương pháp đa thiết kế [1, 2, 3]. Mỗi bản sao là một thành viên trong một nhóm các đối tượng có giao diện giống nhau nhưng có thể được thiết kế bằng các phương pháp khác nhau. Mỗi bản sao có một tham chiếu bản sao IOR (Interoperable Object Reference) duy nhất. Các tham chiếu IOR của các bản sao trong một nhóm đối tượng được tích hợp với nhau tạo thành một bảng tham chiếu gọi là IOGR. Mô hình có thể sử dụng các kiểu nhân bản khác nhau nhóm thành hai loại chính như sau: thụ động - passive và chủ động - active. Sự khác nhau giữa các kiểu nhân bản này là ở thời điểm mà các thành viên của một nhóm đối tượng đạt được trạng thái nhất quán và các cơ chế để đạt được sự nhất quán đó.



Hình 2. Bảng tham chiếu IOGR

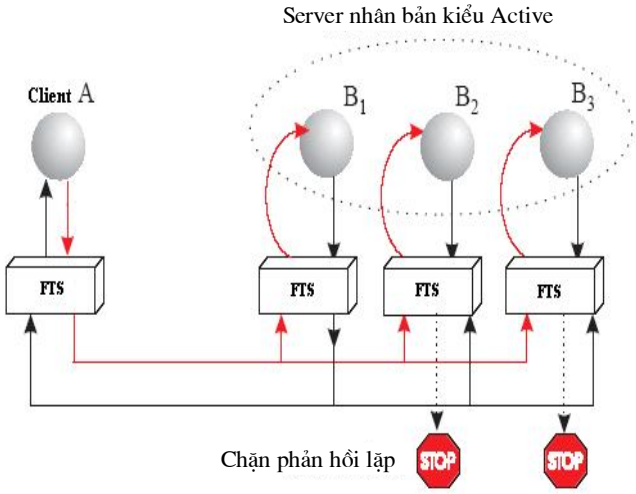
• Kiểu nhân bản thụ động



Hình 3. Kiểu nhân bản thụ động

Với kiểu nhân bản thụ động một bản sao sẽ được chỉ định là bản sao chính trong nhóm các bản sao (xem Hình 3). Bản sao này sẽ có nhiệm vụ thực hiện tất cả các tác vụ được yêu cầu. Các bản sao khác là các bản sao dự phòng sẽ không tiếp nhận yêu cầu và đưa ra phản hồi nào đối với máy khách khi bản sao chính đang hoạt động bình thường. Mục đích của các bản sao dự phòng là để thay thế làm bản sao chính mới khi bản sao chính cũ bị hỏng. Thực tế có nhiều kiểu nhân bản passive, chúng phân biệt với nhau bởi mức độ tụt hậu trạng thái của các bản sao dự phòng so với bản sao chính.

• Kiểu nhân bản chủ động



Hình 4. Kiểu nhân bản chủ động

Với kiểu nhân bản chủ động, tất cả các bản sao đều hoạt động. Khi một client gửi yêu cầu dịch vụ, yêu cầu đó sẽ được chuyển đến tất cả các server bản sao. Tất cả các server bản sao đó sẽ thực hiện tác vụ và trả về phản hồi đến client. Cơ chế truyền thông trong hệ thống phải đảm bảo tất cả các server bản sao sẽ nhận được các yêu cầu giống nhau theo cùng một trật tự giống nhau. Do vậy, các server bản sao sẽ đều thực hiện các tác vụ theo cùng một trật tự. Điều này sẽ đảm bảo trạng thái của các server bản sao là nhất quán vào thời điểm kết thúc của mỗi tác vụ. Trong Hình 4 ta có 3 server bản sao nhân bản kiểu chủ động. Với một yêu cầu của client sẽ được 3 server bản sao xử lý và phản hồi. Như vậy client sẽ nhận được ba phản hồi lặp cho mỗi một yêu cầu. Để đảm bảo tính nhất quán thì những phản hồi lặp như vậy cần được phát hiện và ngăn chặn để đảm bảo chỉ có một phản hồi được đưa đến client. Để giải quyết vấn đề này ta có thể sử dụng cơ chế phát hiện và ngăn chặn phản hồi lặp từ phía nguồn. Khi mỗi server bản sao đưa ra một phản hồi thì hệ thống chịu lỗi sẽ quảng bá phản hồi đó không chỉ đến client mà còn đến tất cả các server bản sao khác. Các server bản sao khác khi nhận được phản hồi này sẽ nhận ra đó là thông điệp "cùng loại", báo yêu cầu đã được phản hồi và do vậy sẽ ngừng không gửi phản hồi đi tiếp. Cũng có thể ngăn chặn phản hồi lặp ở phía đích. Khi các phản hồi được gửi đến phía client, hệ thống phía client sẽ lựa chọn trong số đó một phản hồi và gửi đến ứng dụng client, các phản hồi còn lại sẽ bị bỏ qua.

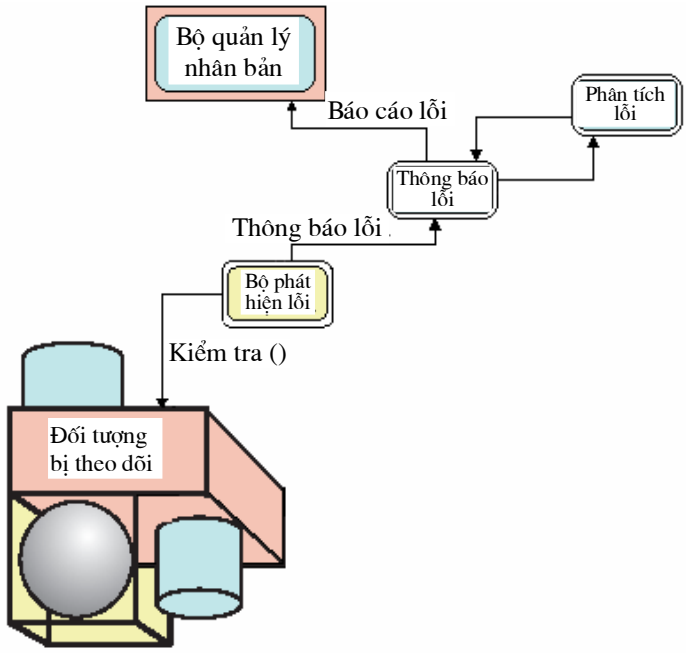
Chi phí sử dụng kiểu nhân bản được xác định bởi các yêu cầu đặc thù của từng hệ thống cụ thể, chẳng hạn như số lượng các bản sao, độ lớn thông tin trạng thái và chu kỳ kiểm tra.

Kiểu nhân bản chủ động được sử dụng nếu chi phí của các thông điệp quảng bá và chi phí xử lý nhân bản ít hơn chi phí truyền trạng thái trong hệ thống. Trong những trường hợp còn lại, kiểu nhân bản thụ động sẽ tốn ít chi phí hơn.

2.2.2. Bộ quản lý nhân bản

Bộ quản lý nhân bản là thành phần khá quan trọng trong các thành phần tạo nên hạ tầng chịu lỗi của mô hình. Trong mô hình này, bộ quản lý nhân bản tương tác với các thành phần khác trong mô hình, tạo ra các nhóm đối tượng, quản lý thuộc tính nhóm đối tượng, điều khiển mối quan hệ giữa các thành viên trong nhóm. Cơ chế hoạt động của nó dựa trên kiểu nhân bản mà hệ thống sử dụng. Thông qua bộ quản lý nhân bản, nhà phát triển có thể xác lập các cấu hình, thuộc tính chịu lỗi cho hệ thống. Có thể thấy nhiệm vụ và vai trò của bộ quản lý nhân bản là rất quan trọng và là một trong những thành phần chính trong mô hình phần mềm chịu lỗi.

2.2.3. Bộ quản lý lỗi



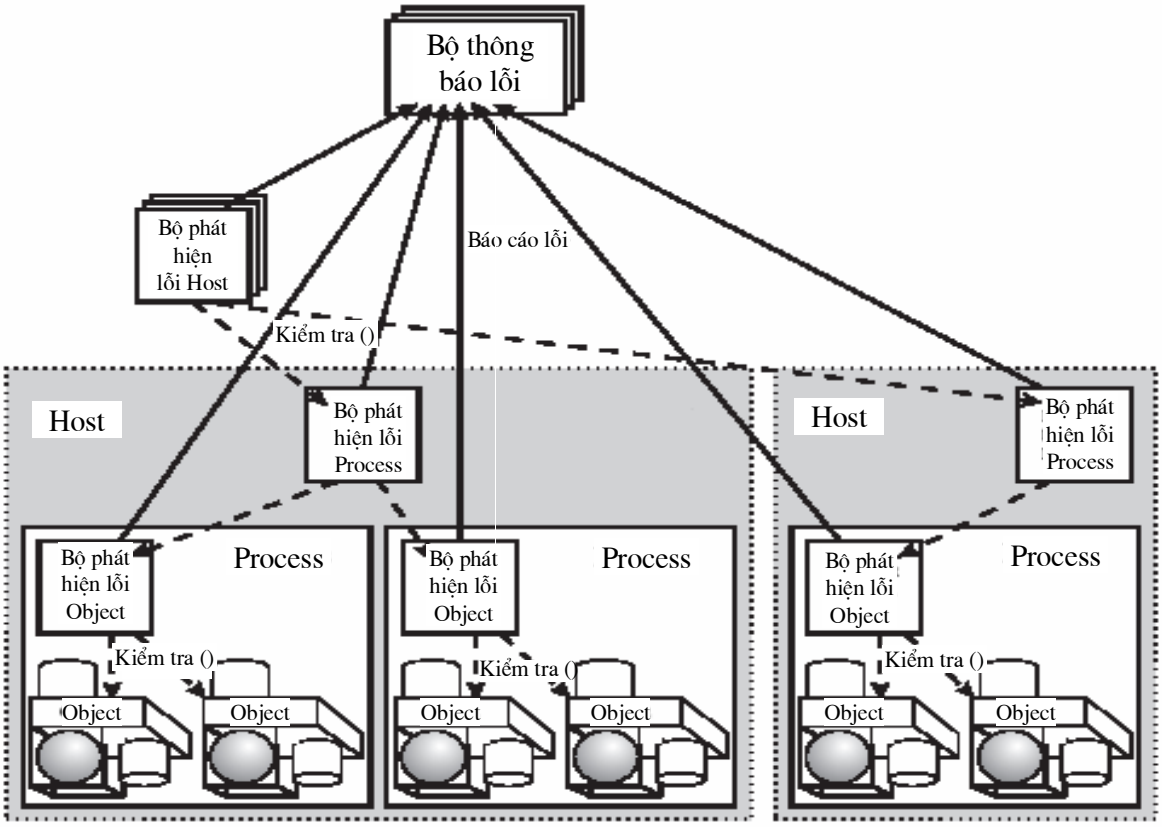
Hình 5. Hoạt động của bộ quản lý lỗi

Trong mô hình phần mềm chịu lỗi này bộ quản lý lỗi được thiết kế bao gồm ba thành phần (xem Hình 5): bộ phát hiện lỗi, bộ thông báo lỗi và bộ phân tích lỗi. Bộ phát hiện lỗi có nhiệm vụ phát hiện sự hiện diện của lỗi trong hệ thống và sinh ra các thông báo lỗi. Bộ thông báo lỗi có nhiệm vụ nhận các thông báo lỗi và gộp lại thành các báo cáo lỗi rồi truyền đến các thành phần đã đăng ký nhận chúng, chẳng hạn như bộ phân tích lỗi và bộ quản lý nhân bản. Bộ phân tích lỗi có nhiệm vụ phân tích các báo cáo lỗi. Dựa trên các báo cáo đó và những thông tin đặc thù của hệ thống, bộ phân tích lỗi sẽ đưa ra quyết định xử lý.

• Bộ phát hiện lỗi

Mô hình hệ thống phát hiện lỗi được xây dựng tương tự như mô hình FT-CORBA, được

mô tả trong Hình 6. Cơ chế theo dõi phát hiện lỗi được xây dựng dựa trên cơ chế timeout, sử dụng kiểu theo dõi Pull hoặc Push. Trong một hệ thống chịu lỗi, số lượng của các bộ phát hiện lỗi là không giới hạn và được tổ chức theo cấu trúc cây phân cấp. Các đối tượng được theo dõi bởi bộ phát hiện lỗi mức đối tượng (object-level Fault Detector). Các bộ phát hiện lỗi mức đối tượng này lại được theo dõi bởi các bộ phát hiện lỗi mức tiến trình (process-level Fault Detector). Các bộ phát hiện lỗi mức tiến trình lại được theo dõi bởi các bộ phát hiện lỗi mức host (host-level Fault Detector). Cấu trúc phát hiện lỗi phân cấp này sẽ tăng hiệu quả chịu lỗi lên rất nhiều và có thể phát hiện lỗi một cách đa dạng.



Hình 6. Cơ chế phát hiện lỗi đa mức

• Bộ phân tích lỗi

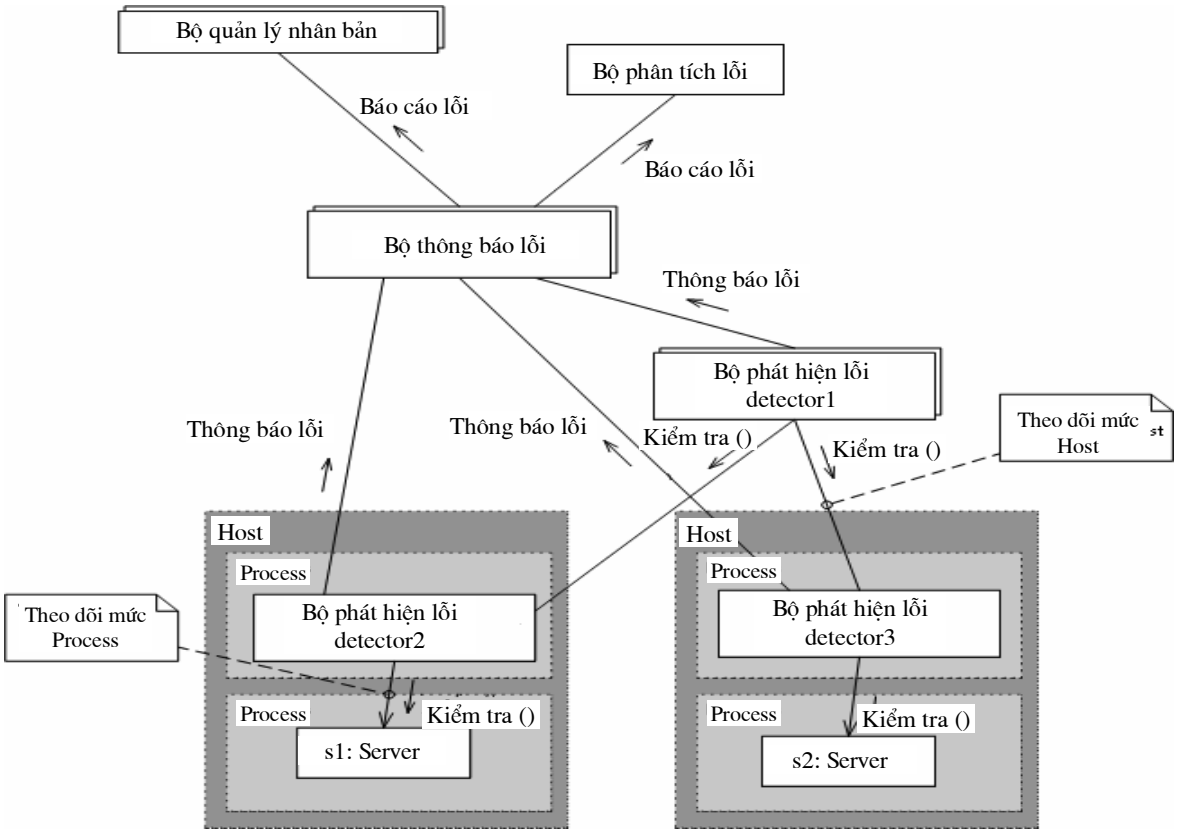
Bộ phân tích lỗi có nhiệm vụ phân tích các báo cáo lỗi, quyết định một thể hiện nào đó của đối tượng mà bộ phát hiện lỗi báo cáo lên có phải là một lỗi hay không? và với lỗi đó thì hệ thống sẽ xử lý như thế nào? Sự xuất hiện của bộ phân tích lỗi là khá quan trọng trong một số hệ thống có những đặc thù riêng và cách thiết kế bộ phân tích lỗi là tùy thuộc vào những đặc thù đó.

• Xử lý thông báo lỗi

Bộ thông báo lỗi chịu trách nhiệm nhận báo cáo lỗi từ các bộ phát hiện lỗi, có thể lọc các báo cáo lỗi để tránh trùng lặp. Sau đó, bộ phận này sẽ gửi các thông báo lỗi đến các đối tượng đã đăng ký nhận chúng để các đối tượng đó xử lý. Ở đây các đối tượng đăng ký nhận

có thể là bộ quản lý nhân bản và bộ phân tích lỗi. Các ứng dụng cũng có thể đăng ký để nhận thông báo lỗi và xử lý theo cách riêng của mình.

2.2.4. Cơ chế lưu thông tin và khắc phục lỗi



Hình 7. Cơ chế lưu thông tin và khắc phục lỗi

Mô hình chịu lỗi trên có một cơ chế lưu thông tin và khắc phục lỗi. Khi hệ thống đang hoạt động bình thường, cơ chế lưu thông tin sẽ ghi lại các thông tin về trạng thái và các tác vụ của các bản sao trong hệ thống. Khi hệ thống bị lỗi, cơ chế khắc phục lỗi sẽ lấy các thông tin đã được ghi này và dùng chúng để khôi phục lại hệ thống bằng cách thiết lập thông tin đó cho các bản sao dự phòng. Nhờ đó, các bản sao dự phòng đạt được trạng thái nhất quán và có thể tiếp tục cung cấp các dịch vụ mà bản sao chính trước đó đã và đang cung cấp. Khi hoạt động, cơ chế truyền thông sẽ quảng bá các thông điệp yêu cầu từ client đến tất cả các server ở các vị trí khác nhau. Tuy nhiên chỉ hệ thống chịu lỗi của server chính nhận thông điệp đó và chuyển cho server chính để xử lý. Hệ thống chịu lỗi ở các server dự phòng sẽ chỉ nhận và lưu lại chứ không xử lý (trường hợp kiểu nhân bản passive). Các thông tin trạng thái và thông điệp truyền đến sẽ được ghi lại trong một log file. Các thông tin này được lưu đúng theo thứ tự mà các bản sao nhận hoặc gửi để sau này ta có thể khắc phục một cách chính xác và nhất quán khi hệ thống bị lỗi. Cơ chế khắc phục lỗi sẽ làm nhiệm vụ đọc và xử lý các thông tin trong log file sau đó sử dụng các thông tin này để thiết lập trạng thái các bản sao trong các trường hợp: thiết lập trạng thái cho replia vừa được phục hồi sau khi

lỗi, thiết lập trạng thái cho bản sao chính mới khi bản sao chính cũ bị lỗi và thiết lập trạng thái cho một bản sao mới vừa được tạo ra. Cần lưu ý trạng thái lưu trong log file có thể bị tụt hậu do chu kỳ lưu thông tin trạng thái lớn và trong chu kỳ đó đối tượng bị lỗi có thể đã nhận và xử lý một số yêu cầu và do đó đã thay đổi trạng thái. Hoặc trong quá trình khắc phục lỗi, hệ thống có thể vẫn tiếp tục nhận được các yêu cầu mới và những yêu cầu đó được lưu giữ lại. Do đó sau khi thiết lập trạng thái xong, hệ thống phải tiến hành gửi lại các yêu cầu đó đến đối tượng thay thế hoặc đối tượng vừa được khắc phục lỗi để đảm bảo tính nhất quán cho hệ thống.

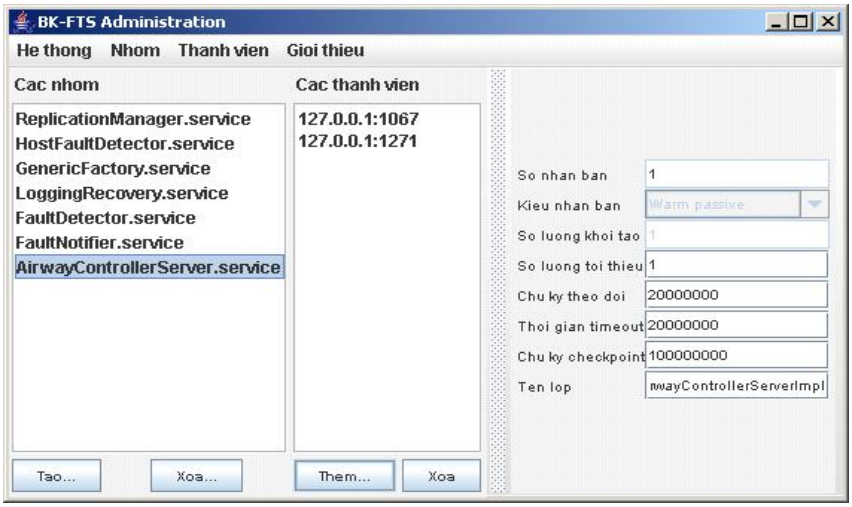
2.2.5. Bộ chọn kết quả

Khi hệ thống phát sinh lỗi, lỗi đó có thể khiến hệ thống đưa ra kết quả sai. Để giải quyết vấn đề này ta sử dụng nhiều phiên bản phần mềm được thiết kế khác nhau cùng xử lý yêu cầu và đưa ra phản hồi. Bộ chọn kết quả trong mô hình sẽ có nhiệm vụ đón nhận tập thông tin phản hồi đó và lựa chọn đưa ra một kết quả "đúng nhất" dựa theo một giải thuật lựa chọn. Việc xây dựng bộ chọn kết quả sẽ tăng cường khả năng chịu lỗi của hệ thống đối với những hành vi ứng xử sai so với yêu cầu do một nguyên nhân bất kỳ nào đó gây lỗi trong hệ thống. Giải thuật lựa chọn sử dụng trong bộ chọn kết quả được thiết kế như thế nào là tùy thuộc vào đặc điểm của từng hệ thống riêng biệt. Một số giải thuật cơ bản bao gồm [3, 4, 7]:

- Các giải thuật quyết định: Exact Majority, Median, Mean.
- Các giải thuật kiểm tra: kiểm tra thỏa mãn yêu cầu; kiểm tra tính hợp lý.

3. THỬ NGHIỆM ĐÁNH GIÁ

3.1. Mô tả bài toán thử nghiệm



Hình 8. Giao diện chính phần mềm thử nghiệm xây dựng theo mô hình BK-FTS

Bài toán xây dựng phần mềm mô phỏng hệ thống kiểm soát không lưu là bài toán điển hình trong lĩnh vực hàng không cho phép các hệ thống trên không và các trạm điều khiển dưới mặt đất liên lạc và trao đổi thông tin với nhau. Hệ thống đòi hỏi cần phải có độ tin cậy và tính chịu lỗi cao. Nếu hệ thống hoạt động không bình thường, đưa ra quyết định sai sẽ

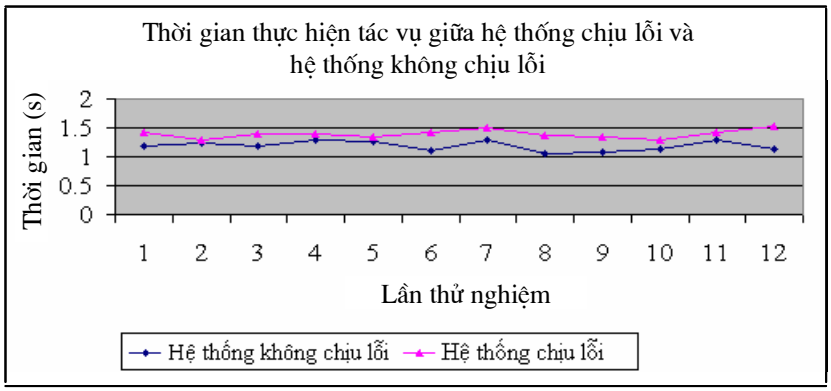
dẫn đến những thiệt hại khôn lường về người và tài sản. Bài toán này cũng được lựa chọn để xây dựng các thử nghiệm trong [2, 10]. Vì vậy chúng tôi lựa chọn bài toán này để thử nghiệm mô hình phần mềm chịu lỗi vào hệ thống với mục đích nghiên cứu hạn chế những lỗi phát sinh bất thường có thể xảy ra trong hệ thống. Tuy nhiên ở đây chúng tôi đặt mục đích thử nghiệm mô hình là chính cho nên không đi sâu vào nghiệp vụ nên hệ thống chỉ mang tính mô phỏng. Chúng tôi sử dụng công cụ lập trình Java 1.2. Hệ thống được thử nghiệm trên hệ điều hành Windows NT Server, các máy trạm sử dụng hệ điều hành Windows XP 2003. Giao diện chính quản lý hệ thống được mô tả ở Hình 8.

3.2. Thử nghiệm

Ở đây chúng tôi triển khai hệ thống đã xây dựng theo mô hình chịu lỗi BK-FTS (Hình 1), sử dụng các giải thuật nhân bản khác nhau và đánh giá hiệu quả của hệ thống thông qua ba thử nghiệm.

Thử nghiệm thứ nhất: Thử nghiệm khả năng chịu lỗi của hệ thống bằng cách tiến hành loại bỏ (đánh sập) một host hoặc gây lỗi một dịch vụ. Kết quả cho thấy hệ thống đã tự động phát hiện và tiến hành khắc phục lỗi đồng thời vẫn tiếp tục cung cấp dịch vụ. Bộ nhớ chiếm dụng: Hệ thống thông thường: 369 (KB); hệ thống chịu lỗi (theo mô hình BK-FTS): 1.875 (KB).

Thử nghiệm thứ hai: Thực hiện 1000 lần một tác vụ, lấy tổng thời gian chia trung bình. Tiến hành 12 lần với mỗi thử nghiệm. Đánh giá thời gian thực hiện một tác vụ trong điều kiện hoạt động bình thường giữa một hệ thống chịu lỗi và một hệ thống không chịu lỗi. Hệ thống không chịu lỗi: 1.184 (s); hệ thống chịu lỗi: 1.412 (s) (Hình 9).



Hình 9. Kết quả thử nghiệm thứ hai

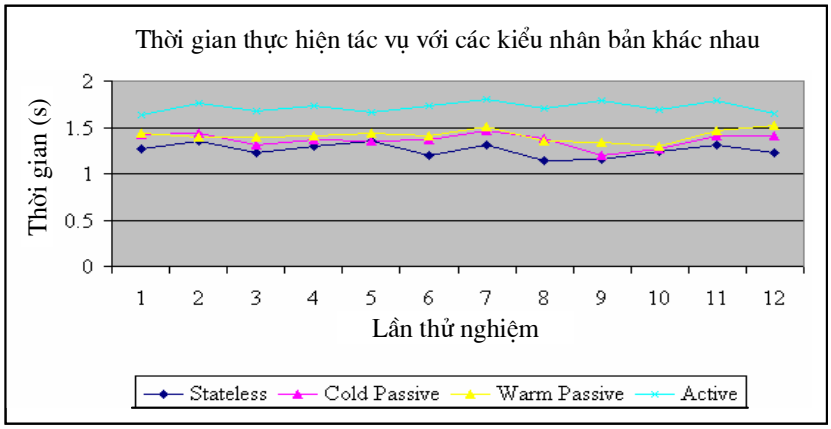
Thử nghiệm thứ ba: Đánh giá thời gian thực hiện một tác vụ trong hệ thống chịu lỗi sử dụng các kiểu nhân bản khác nhau. Thực hiện các thử nghiệm tương tự để đánh giá các kỹ thuật nhân bản khác nhau: Stateless: 1.255 (s); Warm passive:1.412 (s); Cold passive: 1.365 (s); Active: 1.716 (s) (Hình 10).

3.3. Đánh giá kết quả

Thử nghiệm thứ nhất: Kết quả cho thấy hệ thống đã tự động phát hiện và tiến hành khắc phục lỗi đồng thời vẫn tiếp tục cung cấp dịch vụ. Như vậy mô hình phần mềm chịu lỗi mà ta đề xuất đã hoạt động đúng, chứng minh được tính chịu lỗi của phần mềm áp dụng mô hình BK-FTS.

Thử nghiệm thứ hai: Với một hệ thống không chịu lỗi, thời gian thực hiện tác vụ liên quan đến thời gian xử lý tác vụ và băng thông mạng. Đối với hệ thống chịu lỗi, thời gian thực hiện tác vụ còn liên quan đến tài nguyên hệ thống bị chiếm dụng bởi các thành phần cung cấp khả năng chịu lỗi cho phần mềm. Do vậy thời gian hoàn thành một tác vụ của hệ thống chịu lỗi sẽ chậm hơn hệ thống không chịu lỗi. Kết quả thử nghiệm đã thể hiện điều đó. Tuy nhiên ta cũng thấy rằng thời gian chêch lệch là không nhiều và chấp nhận được (xem Hình 9).

Thử nghiệm thứ ba: Thời gian hoàn thành tác vụ với từng kiểu nhân bản được sắp xếp theo thứ tự tăng dần như sau: stateless, cold passive, warm passive, active. Như đã phân tích trong phần so sánh các kiểu nhân bản, nói chung các kiểu nhân bản thụ động thường tốn ít tài nguyên hơn so với kiểu nhân bản chủ động. Kiểu nhân bản chủ động đòi hỏi có nhiều bản sao cùng hoạt động dẫn đến phải chi phí nhiều hơn cho các thông điệp quảng bá và xử lý nhân bản (xem Hình 10).



Hình 10. Kết quả thử nghiệm thứ ba

3.4. Một số nhận xét và so sánh

Như trên ta đã trình bày về một số mô hình phần mềm chịu lỗi phát triển cho hệ phân tán trên thế giới bao gồm ba mô hình: mô hình AFT-CCM, mô hình FT-CORBA, mô hình dựa trên CORBA Trading Service. Trong bài báo, chúng tôi đã trình bày các thành phần của mô hình phần mềm chịu lỗi BK-FTS, giải thích cơ chế hoạt động của từng thành phần, thực hiện các đánh giá thử nghiệm. Tiếp theo, chúng ta so sánh mô hình BK-FTS với các mô hình trên theo các tiêu chí: khả năng phát hiện lỗi, khả năng phân tích lỗi, cơ chế khắc phục lỗi.

So sánh về khả năng phát hiện lỗi: Các thử nghiệm đã cho thấy mô hình BK-FTS có khả năng chịu lỗi bất kỳ bao gồm các lỗi mức đối tượng (đối tượng ngừng cung cấp dịch vụ), lỗi mức host (một host cụ thể ngừng cung cấp dịch vụ trên đó) và lỗi dữ liệu (một dịch vụ cung cấp sai về nội dung) nhờ các bộ phát hiện lỗi và bộ chọn kết quả. Ngoài ra, trong mô hình ta xây dựng tổ chức phát hiện lỗi phân cấp giúp nâng cao khả năng và hiệu quả chịu lỗi của hệ thống. Các mô hình khác không toàn vẹn như vậy, mô hình AFT-CCM không đề cập đến khả năng chịu lỗi dữ liệu, không tổ chức phát hiện lỗi phân cấp, mô hình FT-CORBA không chịu lỗi dữ liệu, mô hình dựa Trading Service không đề cập đến khả năng phát hiện lỗi [2, 8, 9, 10].

So sánh về khả năng phân tích lỗi: Mô hình BK-FTS cung cấp bộ phân tích lỗi cho phép thực

hiện một giải thuật phân tích lỗi cụ thể tùy theo từng hệ thống đặc thù. Sự xuất hiện của bộ phân tích lỗi là khá quan trọng bởi trong một số hệ thống thì một số lỗi mà bộ phát hiện lỗi phát hiện ra cần được xử lý theo đặc thù riêng của hệ thống đó. Cách thiết kế và giải thuật thực thi bộ phân tích lỗi là tùy thuộc vào những đặc thù đó. Các mô hình khác không đề cập đến vấn đề này.

So sánh về cơ chế khắc phục lỗi: Mô hình BK-FTS cung cấp cơ chế lưu thông tin và khắc phục lỗi làm nhiệm vụ ghi chép các yêu cầu đến, các phản hồi lại và các điểm đánh dấu hệ thống checkpoint của các đối tượng nhân bản nhằm phục vụ cho mục đích khắc phục về sau. Mô hình FT-CORBA cũng cung cấp cơ chế này, tuy nhiên mô hình AFT-CCM lại không tách bạch rõ ràng cơ chế đó, mô hình dựa Trading Service cũng không tách bạch rõ ràng.

Đánh giá chung: Như vậy, có thể kết luận mô hình BK-FTS đã hỗ trợ đầy đủ các tính năng chịu lỗi, các mô hình khác thì không hỗ trợ hoặc không thiết kế rõ ràng chức năng nhiệm vụ. Mô hình BK-FTS đáp ứng đầy đủ yêu cầu của một mô hình phần mềm chịu lỗi. Với mô hình BK-FTS, các nhà phát triển ứng dụng hoàn toàn có thể áp dụng cho các hệ thống yêu cầu độ tin cậy cao với những tiêu chí về tài nguyên, kinh tế, tính khả thi. Có thể nói mô hình BK-FTS đã cung cấp một giải pháp hoàn chỉnh, có ý nghĩa khoa học cũng như thực tiễn cao, phù hợp với nhu cầu thực tế.

Xây dựng phần mềm chịu lỗi cần chú ý đến lựa chọn các thành phần để thực hiện các kỹ thuật chịu lỗi và lựa chọn các kỹ thuật chịu lỗi cụ thể áp dụng trong từng thành phần. Trong thực tế chi phí để thực hiện kỹ thuật chịu lỗi khá cao, do đó việc lựa chọn phần nào trong toàn hệ thống để áp dụng kỹ thuật chịu lỗi là việc rất quan trọng. Một điểm nữa là lựa chọn kiểu nhân bản nào để áp dụng cho bài toán đặt ra. Điều này thường dựa trên những phân tích về tính hiệu quả/chí phí, dựa trên thực tế kinh nghiệm của từng người để lựa chọn kiểu nhân bản phù hợp. Số biến thể chương trình (bản sao) cũng là một việc quan trọng đòi hỏi người xây dựng phần mềm chịu lỗi phải quan tâm đến. Càng nhiều biến thể chương trình thì càng tăng thời gian thực hiện của toàn hệ thống và tất nhiên cũng tăng chi phí xây dựng hệ thống. Cuối cùng, cần lựa chọn thuật toán hiệu quả để xây dựng các thành phần như bộ phát hiện lỗi, bộ phân tích lỗi, bộ chọn kết quả trong mô hình. Các kỹ thuật trên được lựa chọn như thế nào cho phù hợp với bài toán cụ thể sẽ phụ thuộc vào chi phí của dự án và kinh nghiệm của đội ngũ phát triển hệ thống.

4. KẾT LUẬN VÀ HƯỚNG NGHIÊN CỨU TIẾP THEO

Bài báo đã trình bày về tư tưởng, một số phương pháp xây dựng phần mềm chịu lỗi, mô hình phần mềm chịu lỗi, cơ chế hoạt động, các thành phần của mô hình phần mềm chịu lỗi BK-FTS. Các tác giả xây dựng một hệ thống phần mềm mô phỏng kiểm soát không lưu áp dụng mô hình BK-FTS đề xuất nhằm thử nghiệm tính đúng đắn của mô hình. Các tác giả tiến hành đánh giá khả năng chịu lỗi của hệ thống, các giải thuật nhân bản khác nhau và trên cơ sở đó so sánh đánh giá mô hình phần mềm BK-FTS với các mô hình phần mềm chịu lỗi khác theo các tiêu chí khả năng phát hiện lỗi, khả năng phân tích lỗi và khả năng khắc phục lỗi.

Để phát triển một hệ thống phần mềm chịu lỗi, chúng tôi thấy cần phải quan tâm đặc biệt đến các vấn đề sau:

- Tăng cường tối đa tính tin cậy của từng thành phần tham gia vào hệ thống.

• Xác định mức độ chịu lỗi mà hệ thống muốn áp dụng, từ đó lựa chọn kiểu nhân bản phù hợp, số lượng bản sao sử dụng trong hệ thống cùng một số tham số chịu lỗi khác.
• Nghiên cứu các giải thuật nhằm xây dựng các bộ phát hiện lỗi, bộ phân tích lỗi, bộ chọn kết quả một cách hiệu quả dựa trên đặc thù của từng hệ thống và kinh nghiệm thực tế.

Các nội dung này hiện đang được chúng tôi tập trung nghiên cứu.

TÀI LIỆU THAM KHẢO

[1] Huỳnh Quyết Thắng, Phạm Bá Quang, Lương Thanh Bình, So sánh đánh giá các kỹ thuật xây dựng phần mềm chịu lỗi, *Kỷ yếu Hội thảo khoa học quốc gia về Nghiên cứu phát triển và ứng dụng Công nghệ thông tin và truyền thông ICT.RDA lần thứ II*, Hà Nội 24-25/9 (175–185).

[2] Priya Narasimhan, Transparent Fault Tolerance for CORBA, *Dissertation for the Degree of Doctor of Philosophy in Electrical and Computer Engineering*, University of California, 1999. http://www.cs.cmu.edu/∼priya/papers/priya-PhD-1999.pdf

[3] F. Di Giandomenico, A. Bondavalli, J. Xu, and S. Chiaradonna, An adaptive approach to achieving hardware and software fault tolerance in a distributed computing environment, *JSA - Journal on Systems and Architectures* No. 2 (2001) 102–110.

[4] P. E. Ammann, Knight J. C. Data Diversity, An approach to software fault tolerance, *IEEE Transactions on Computers* Vol. C-37, No. 4 (1988) 418–425.

[5] R. Baldoni, J. M. Helary, and M. Raynal, From crash fault-tolerance to arbitrary-fault tolerance: Towards a modular approach, *Proceedings of the International Conference on Dependable Systems and Networks*, New York, USA, June 25-28, 2000 (273–282).

[6] C. S. Wu, Formal specification techniques and their applications in N-version programming, *Dissertation for the Degree of Doctor of Philosophy, UCLA*, Computer Science Department, October 1990.
http://genealogy.math.ndsu.nodak.edu/html/id.phtml?id=70897

[7] P. Popov, L. Strigini, A. Kostov, V. Mollov, and D. Selensky, Software fault-tolerance with off-the-shelf SQL servers, *Proc. 3rd International Conference on Component-Based Software Systems (ICCBSS'04)*, Redondo Beach, CA, U.S.A., Feb. 2-4, 2004 (117–126).

[8] Fábio Favarim, Frank Siqueira, Joni Fraga, *Adaptive Fault-tolerant CORBA Components* Departament of Computer Science Federal University of Santa Catarina - Florianópolis, SC 88040-900 - Brazil.
http : //www.das.ufsc.br/∼fabio/www/papers/middleware2003.pdf .

[9] René Meier, "A framework providing fault tolerance using the CORBA trading service", A Master thesis, University of Dublin, Department os Computer Science, 1998.

[10] https://www.cs.tcd.ie/publications/ tech-reports/reports.99/TCD-CS-1999-31.pdf

[11] ObjectManagementGroup, CommonObject Request Broker Architecture: Core Specification Version 3.0.2 - Editorial update formal/02-12-02. December 2002.
http://www.omg.org/technology/documents/corba_spec_catalog.htm

Nhận bài ngày 24 - 6 - 2005
Nhận lại sau sửa ngày 5 - 1 - 2006