Tạp chí Tin học và Điều khiển học, T.25, S.3 (2009), 333–344

ĐÁNH GIÁ ĐỘ SẴN SÀNG CỦA PHẦN MỀM CHỊU LỖI ÁP DỤNG KỸ THUẬT REJUVENATION VÀ NVP HAI PHIÊN BẢN

HUỲNH QUYẾT THẮNG, VŨ HỒNG LĨNH

Khoa Công nghệ thông tin, Đại học Bách khoa Hà Nội

Abstract. Rejuvenation and NVP are two well-known software fault-tolerant techniques. They are based on two different concepts: design diversity and execution environment diversity. Previous researchs on these techniques mainly focus on evaluation and analysis of each approach separately. In this paper, we propose an approach for utilizing advantages of two techniques in order to improve fault-tolerant systems availability, especially continuously running system. Using continuous Markov chains, we evaluate the availability of two versions hot-standby system applying proposed model, taking into account correlation between two versions. Some experimental results to examine the reasonableness os the approach also are presented.

Tóm tắt. Kỹ thuật Rejuvenation và kỹ thuật NVP là hai kỹ thuật xây dựng phần mềm chịu lỗi tiêu biểu, đại diện cho hai nhóm phương pháp dựa trên sự đa dạng trong môi trường thực thi và sự đa dạng trong môi trường phát triển. Các nghiên cứu trước kia về hai kỹ thuật này thường chỉ áp dụng và đánh giá từng kỹ thuật một cách riêng lẻ. Trong bài báo này, chúng ta thử nghiệm mô hình kết hợp hai kỹ thuật này nhằm nâng cao độ sẵn sàng cho các hệ thống phần mềm chịu lỗi, đặc biệt là các hệ thống hoạt động liên tục theo thời gian. Sử dụng mô hình Markov, chúng tôi đánh giá độ sẵn sàng của hệ thống dự phòng nóng hai phiên bản áp dụng mô hình kết hợp đã đề xuất, trong đó có xem xét đến sự tương quan giữa hai phiên bản. Một số kết quả thực nghiệm chứng minh tính hợp lý của mô hình cũng được trình bày.

Keyword: software fault-tolerant, rejuvenation, NVP, availability, software aging

1. MỞ ĐẦU

Trong thời gian gần đây, nhiều nghiên cứu về các hệ thống hoạt động và cung cấp dịch vụ liên tục theo thời gian đã chỉ ra hiện tượng lão hóa (Software Agging) trong các hệ thống phần mềm [1, 3, 4] khi phần mềm hoạt động liên tục theo thời gian, một số sai sót sẽ làm cho hệ thống bị lão hóa, thể hiện qua việc tài nguyên bị rò rỉ, hiệu năng hệ thống giảm [1, 3, 4, 5]. Huang đã nghiên cứu về hiện tượng này trong hệ thống thanh toán truyền thông [1], trong đó ứng dụng theo thời gian sẽ gặp phải hỏng hóc hay ngưng hoạt động. Huang đề xuất phương pháp mới, mang tính phòng ngừa được gọi là bảo trì phòng ngừa (Rejuvenation) để giải quyết vấn đề lão hóa phần mềm [1]. Ý tưởng cơ bản của phương pháp này là theo định kỳ dừng hệ thống đang hoạt động liên tục, dọn dẹp tài nguyên hệ thống sau đó khởi động lại hệ thống nhằm đưa hệ thống về trạng thái hoạt động tốt nhất ban đầu [1, 2].

Có hai hướng nghiên cứu trong lĩnh vực nghiên cứu về lão hóa phần mềm [1 – 7]. Hướng nghiên cứu dựa trên thực nghiệm tập trung vào xác định và kiểm tra sự tồn tại của lão hóa phần mềm và ảnh hưởng của nó lên tài nguyên hệ thống [1, 2, 4]. Hướng còn lại là hướng dựa trên mô hình, trong đó tập trung vào việc đánh giá sự hiệu quả của bảo trì phòng ngừa và đưa là lịch bảo trì phòng ngừa tối ưu [3, 5 – 7].

Huang thực hiện mô hình lão hóa trong hệ thống bằng một quá trình gồm hai bước [1, 4]. Ban đầu hệ thống hoạt động trong trạng thái hoàn toàn bình thường không có lỗi, theo thời gian, hệ thống sẽ rơi vào trạng thái có khả năng bị lỗi và cuối cùng là rơi vào trạng thái bị lỗi. Áp dụng chuỗi Markov liên tục về thời gian, Huang đã đưa ra giới hạn để việc bảo trì phòng ngừa đạt được hiệu quả [1, 2, 4].

Trivedi đề xuất mô hình Rejuvenation sửa đổi, để phù hợp hơn với thực tế, sau đó đề xuất mô hình kết hợp với kỹ thuật NVP [12] để tăng cường độ sẵn sàng cho phần mềm chịu lỗi, tận dụng sự đa dạng trong cả quá trình phát triển và môi trường thực thi. NVP đại diện cho nhóm các kỹ thuật dựa trên sự khác biệt trong thiết kế và phát triển và Rejuvenation đại diện nhóm các kỹ thuật dựa trên sự khác biệt trong môi trường thực thi. Tiếp tục phát triển nghiên cứu theo định hướng này, chúng tôi đi sâu vào thử nghiệm mô hình cho hệ thống dự phòng nóng hai phiên bản, xây dựng một số đánh giá về độ sẵn sàng, cũng như sự phụ thuộc giữa hai phiên bản, ảnh hưởng của nó lên độ sẵn sàng của hệ thống.

Bài báo được cấu trúc như sau. Mục 1, giới thiệu mở đầu. Mục 2 trình bày mô hình kết hợp hai kỹ thuật Rejuvenation và NVP để nâng cao độ sẵn sàng cho phần mềm chịu lỗi. Mục 3 trình bày đánh giá mô hình áp dụng cho hệ thống dự phòng nóng hai phiên bản. Các kết quả thử nghiệm được trình bày trong Mục 4 của bài báo. Và cuối cùng là kết luận và hướng nghiên cứu tiếp.

2. MÔ HÌNH KẾT HỢP REJUVENATION VÀ NVP

2.1. Ý tưởng

Các kỹ thuật xây dựng phần mềm chịu lỗi đều có chung một mục đích là cải thiện được độ tin dùng của hệ thống trên các nguyên lý khác nhau như sự khác biệt về thiết kế hay sự khác biệt về môi trường thực thi.

Hai kỹ thuật này là hai kỹ thuật tiêu biểu cho hai nhóm kỹ thuật xây dựng phần mềm chịu lỗi:

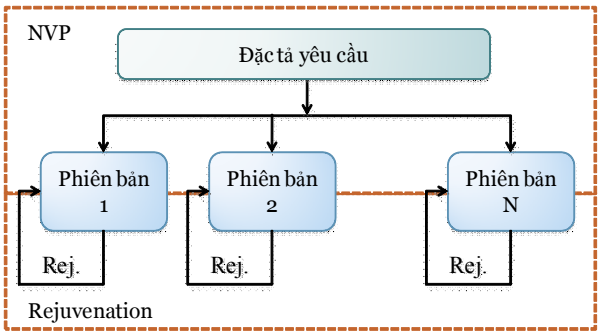
• NVP đại diện cho nhóm các kỹ thuật dựa trên sự khác biệt trong thiết kế và phát triển. Điểm mạnh của kỹ thuật NVP là sự khác biệt trong môi trường thiết kế và phát triển, từ đó giảm thiểu mối sự tương quan về lỗi giữa các phiên bản nhờ việc các phiên bản được phát triển độc lập.

• Rejuvenation đại diện nhóm các kỹ thuật dựa trên sự khác biệt trong môi trường thực thi. Điểm mạnh của kỹ thuật Rejuvenation là nó tạo ra sự khác biệt trong môi trường thực thi của phần mềm, chống lại hiện tượng lão hóa tiến trình nhờ việc chủ động cho tiến trình ngừng hoạt động, dọn dẹp tài nguyên sau đó khởi động lại.

Kết hợp hai kỹ thuật, tận dụng được điểm mạnh của từng kỹ thuật xây dựng phần mềm chịu lỗi, phần mềm sẽ có được cả sự khác biệt về thiết kế, phát triển và và sự khác biệt trong môi trường thực thi, từ đó độ sẵn sàng được nâng cao.

2.2. Mô hình kết hợp

Xuất phát từ ý tưởng được trình bày ở trên, mô hình đề xuất để kết hợp hai kỹ thuật sẽ gồm có hai pha (Hình 1).



Hình 1. Mô hình kết hợp Rejuvenation và NVP

• Pha phát triển: Từ đặc tả yêu cầu ban đầu, tương tự như trong kỹ thuật lập trình đa phiên bản NVP, N phiên bản sẽ được phát triển độc lập bởi các đội phát triển độc lập với các công cụ khác nhau. Các phiên bản được cài đặt riêng biệt nhằm giảm khả năng gặp phải những lỗi giống nhau trong quá trình hoạt động, từ đó tăng cường tính sẵn sàng của hệ thống.

• Pha thực thi: N phiên bản được đưa vào hoạt động song song, tuy nhiên khác với kỹ thuật NVP truyền thống, chúng ta không sử dụng bộ quyết định để kiểm tra kết quả của các phiên bản, mà các phiên bản được chạy song song cùng cung cấp dịch vụ cho người dùng. Mỗi phiên bản đều áp dụng kỹ thuật Rejuvenation nhằm ngăn ngừa tình trạng lỗi do lão hóa tiến trình.

2.3. Áp dụng mô hình cho hệ thống dự phòng nóng

Trong hệ thống dự phòng nóng, với yêu cầu hoạt động và cung cấp dịch vụ liên tục, người ta thường dựa trên nguyên tắc dư thừa về tài nguyên hệ thống để đảm bảo tính sẵn sàng của của hệ thống. Thông thường, một phiên bản của hệ thống được nhân bản và cho hoạt động song song nhằm giảm khả năng gặp sự cố của hệ thống và tăng cường tính sẵn sàng của hệ thống. Như thế, khả năng rất lớn các phiên bản của hệ thống gặp phải những lỗi giống nhau và đồng thời ngừng cung cấp dịch vụ. Thêm vào đó, các hệ thống này được yêu cầu chạy liên tục theo thời gian, nên sẽ xảy ra hiện tượng lão hóa tiến trình trong hệ thống.

Như vậy, chúng ta có thể thấy rằng, hệ thống dự phòng nóng rất thích hợp để áp dụng mô hình đề xuất ở trên nhằm tăng độ sẵn sàng của hệ thống.

Các nghiên cứu trước đây [8 – 12] về các hệ thống chịu lỗi sử dụng phương pháp dư thừa về tài nguyên đã chứng minh rằng, hệ thống chịu lỗi với hai phiên bản sẽ cho hiệu quả cao

nhất. Kế thừa kết quả nghiên cứu đó, chúng ta sẽ sử dụng số phiên bản trong mô hình kết hợp bảo trì phòng ngừa với lập trình đa phiên bản là $N = 2$. Ngoài ra, hai phiên bản này được xây dựng hoàn toàn độc lập, theo kỹ thuật lập trình đa phiên bản NVP. Dựa trên cùng một bản yêu cầu về hệ thống, hai phiên bản sẽ được phân tích, thiết kế và cài đặt độc lập hoàn toàn bởi hai nhóm phát triển khác nhau. Tuy nhiên, khác với kỹ thuật lập trình nhiều phiên bản sẽ không có bộ so sánh kết quả giữa hai phiên bản mà hai phiên bản này sẽ được cho hoạt động song song. Thêm vào đó, mỗi phiên bản sẽ được áp dụng kỹ thuật bảo trì phòng ngừa. Trên cùng quan điểm với mô hình do Huang về bảo trì phòng ngừa chúng ta giả sử rằng mỗi phiên bản sẽ bị thoái hóa trong thời gian hoạt động và sẽ tiến tới trạng thái bị lỗi nếu không có quá trình bảo trì phòng ngừa được kích hoạt khi hệ thống chuyển sang trạng thái có khả năng bị lỗi.

3. PHƯƠNG PHÁP ĐÁNH GIÁ ĐỘ SẴN SÀNG CỦA HỆ THỐNG XÂY DỰNG THEO MÔ HÌNH KẾT HỢP REJUVENATION VÀ NVP HAI PHIÊN BẢN

Một hệ thống dự phòng nóng hai phiên bản, với mỗi phiên bản sẽ có hai mức thoái hóa là có khả năng bị lỗi và bị lỗi. Thời gian chuyển từ trạng thái có khả năng bị lỗi sang trạng thái bị lỗi tuân theo phân phối mũ. Gọi X và Y lần lượt là thời gian thoái hóa của Phiên bản 1 và Phiên bản 2 với hàm phân phối xác suất là:

$$\begin{aligned} F_X(x) &= 1 - e^{-\lambda_1 x},\ x > 0,\ \lambda_1 > 0, \\ F_Y(y) &= 1 - e^{-\lambda_2 y},\ y > 0,\ \lambda_2 > 0. \end{aligned} \tag{1}$$

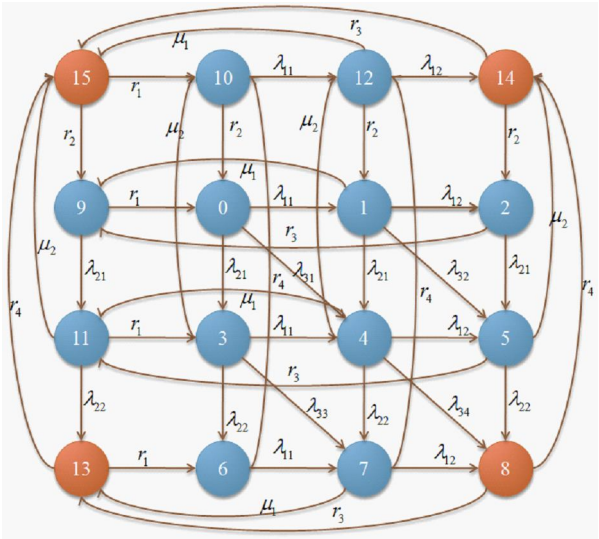
Nếu quá trình thoái hóa dẫn đến trạng thái lỗi của hai phiên bản là độc lập về thống kê, phân phối đồng thời sẽ là:

$$\begin{aligned} F_{X,Y}(x,y) &= P\{X \le x,\ Y \le y\} \\ &= 1 - \{1 - F_X(x)\}\{1 - F_Y(y)\} \\ &= 1 - e^{-\lambda_1 x - \lambda_2 y}. \end{aligned} \tag{2}$$

Trong thực tế, việc xây dựng hai phiên bản hoàn toàn độc lập với cùng chức năng, do vậy chúng ta cần xét đến sự tương quan của quá trình thoái hóa của hai phiên bản. Trong nghiên cứu này chúng ta sẽ sử dụng phân phối mũ hai chiều của Marshall và Olkin [13] để biểu diễn hiện tượng hai phiên bản cùng thoái hóa song song. Việc áp dụng phân phối này nhằm đơn giản hóa việc biểu diễn quá trình thoái hóa đồng thời của hai phiên bản và đảm bảo việc áp dụng chuỗi Markov liên tục về thời gian để phân tích hệ thống. Với thời gian X và Y của hai phiên bản ở trên, thời gian thoái hóa đồng thời của hai hệ thống sẽ tuân theo phân phối mũ hai chiều sau

$$\begin{aligned} F(x,y) &= P\{X \le x,\ Y \le y\} \\ &= 1 - e^{-\lambda_1 x - \lambda_2 y - \lambda_{12} \max(x,y)}. \end{aligned} \tag{3}$$

Giả sử cả hai phiên bản của hệ thống được áp dụng bảo trì phòng ngừa, sử dụng mô hình Rejuvenation do Trivedi đề xuất [4], chúng ta có sơ đồ chuyển trạng thái của xích Markov liên tục về thời gian biểu diễn hệ thống sẽ là:



Hình 2. Lược đồ chuyển trạng thái khi áp dụng Rejuvenation kết hợp với NVP

Các trạng thái tương ứng của hệ thống

Bảng 1. Các trạng thái của hệ thống dự phòng nóng áp dụng mô hình

Trạng thái	Mô tả
0	Cả hai phiên bản hoạt động bình thường không lỗi
1 (3)	**Phiên bản 1 (2) hoạt động bình thường, phiên bản 2 (1) rơi vào trạng thái có thể gây lỗi**
2 (6)	Phiên bản 1 (2) hoạt động bình thường, phiên bản 2 (1) bị lỗi
5 (7)	**Phiên bản 1 (2) rơi vào trạng thái có thể bị lỗi, phiên bản 2 (1) bị lỗi**
9 (10)	Phiên bản 1 (2) bảo trì phòng ngừa, phiên bản 2 (1) hoạt động bình thường
11 (12)	**Phiên bản 1 (2) bảo trì phòng ngừa, phiên bản 2 (1) rơi vào trạng thái có thể bị lỗi**
13 (14)	Phiên bản 1 (2) bảo trì phòng ngừa, phiên bản 2 (1) bị lỗi
4	**Cả hai phiên bản đều rơi vào trạng thái có thể bị lỗi**
8	Cả hai phiên bản đều bị lỗi (hệ thống ngưng hoạt động)
15	**Cả hai phiên bản trong quá trình bảo trì phòng ngừa**

Dựa trên định nghĩa các trạng thái ở trên, ta có hệ thống bị dừng hoạt động sẽ tương đương với các trạng thái $(8, 13, 14, 15)$.

Với các tham số$\lambda_{11}, \lambda_{12}, \mu_1, r_1, r_3$ $(\lambda_{21}, \lambda_{22}, \mu_2, r_2, r_4)$ lần lượt là tốc độ chuyển trạng thái từ hoạt động bình thường sang có thể bị lỗi, tốc độ lỗi từ trạng thái có thể bị lỗi, tốc độ kích hoạt bảo trì phòng ngừa từ trạng thái có thể bị lỗi, tốc độ phục hồi từ khi bảo trì phòng ngừa được kích hoạt, tốc độ hồi phục sau lỗi của phiên bản 1(2). Và $\lambda_{31}, \lambda_{32}$ $(\lambda_{33}), \lambda_{34}$ lần lượt là tốc độ hai phiên bản đồng thời thoái hóa, tốc độ phiên bản 1 (2) lỗi còn phiên bản 2 (1) thoái hóa đồng thời, tốc độ cả hai phiên bản đồng thời bị lỗi từ trạng thái có thể bị lỗi.

Giả sử $\{X(t) = i,\ t \geq 0\},\ i = 0, ..15$ là chuỗi Markov liên tục về thời gian biểu diễn trạng thái thống kê của hệ thống.

Gọi $Q_{0,j}(t) = Pr\{X(t) = j | X(0) = 0\},\ t \geq 0,\ j = 0, ..15$ là xác suất hệ thống ở trạng

thái j tại thời điểm t biết trạng thái ban đầu là 0.

Dựa trên các trạng thái định nghĩa ở trên ta sẽ có được độ sẵn sàng của hệ thống là

$$\begin{aligned}A(t) =&Q_{0,0}(t) + Q_{0,1}(t) + Q_{0,2}(t) + Q_{0,3}(t) + Q_{0,4}(t) + Q_{0,5}(t)\\ &+ Q_{0,6}(t) + Q_{0,7}(t) + Q_{0,8}(t) + Q_{0,9}(t) + Q_{0,10}(t) + Q_{0,11}(t) + Q_{0,12}(t).\end{aligned} \quad (4)$$

Giả sử rằng tồn tại giới hạn $p_j = \lim_{t\to\infty} Q_{0,j}(t),\ j = 0, .., 15$ đó chính là xác suất hệ thống ở trạng thái j ở trạng thái ổn định.

Cho $t \to \infty$ từ phương trình Chapman- Kolmogrov của chuỗi Markov liên tục về thời gian biểu diễn hệ thống ta sẽ thu được hệ phương trình đại số sau

$$-(\lambda_{11} + \lambda_{21} + \lambda_{31})p_0 + rp_9 + r_2p_{10} = 0 \quad (5)$$
$$-(\lambda_{12} + \lambda_{21} + \lambda_{31} + \mu_1)p_1 + \lambda_{11}p_0 + r_2p_{12} = 0 \quad (6)$$
$$-(\lambda_{21} + r_3)p_2 + \lambda_{12}p_1 + r_2p_{14} = 0 \quad (7)$$
$$-(\lambda_{11} + \lambda_{22} + \lambda_{33} + \mu_2)p_3 + \lambda_{21}p_0 + r_1p_{11} = 0 \quad (8)$$
$$-(\lambda_{12} + \lambda_{22} + \lambda_{34} + \mu_1 + \mu_2)p_4 + \lambda_{31}p_0 + \lambda_{21}p_1 + \lambda_{11}p_3 = 0 \quad (9)$$
$$-(\lambda_{22} + r_3)p_5 + \lambda_{32}p_1 + \lambda_{21}p_2 + \lambda_{12}p_4 = 0 \quad (10)$$
$$-(\lambda_{11} + r_4)p_6 + \lambda_{22}p_3 + r_1p_{13} = 0 \quad (11)$$
$$-(\lambda_{12} + r_4)p_7 + \lambda_{33}p_3 + \lambda_{22}p_4 + \lambda_{11}p_6 = 0 \quad (12)$$
$$-(r_3 + r_4)p_8 + \lambda_{34}p_4 + \lambda_{22}p_5 + \lambda_{12}p_7 = 0 \quad (13)$$
$$-(\lambda_{21} + r_1)p_9 + \mu_1p_1 + r_3p_2 + r_2p_{15} = 0 \quad (14)$$
$$-(\lambda_{11} + r_2)p_{10} + \mu_2p_3 + r_4p_6 + r_1p_{15} = 0 \quad (15)$$
$$-(\lambda_{22} + r_1 + \mu_2)p_{11} + \mu_1p_4 + \lambda_{21}p_9 + r_3p_5 = 0 \quad (16)$$
$$-(\lambda_{12} + r_2 + \mu_1)p_{12} + \mu_2p_4 + \lambda_{11}p_{10} + r_4p_7 = 0 \quad (17)$$
$$-(r_1 + r_4)p_{13} + \lambda_{21}p_{11} + \mu_1p_7 + r_3p_8 = 0 \quad (18)$$
$$-(r_2 + r_3)p_{14} + \lambda_{12}p_{12} + \mu_2p_5 + r_4p_8 = 0 \quad (19)$$
$$-(r_1 + r_2)p_{15} + \mu_2p_{11} + r_4p_{13} + \mu_1p_{12} + r_3p_{14} = 0 \quad (20)$$
$$\sum_{j=0}^{15} p_j = 1. \quad (21)$$

Giải phương trình đại số trên ta sẽ thu được phân bố của hệ thống ở trạng thái ổn định và độ sẵn sàng của hệ thống khi đó là

$$A = p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 + p_9 + p_{10} + p_{11} + p_{12}. \quad 22$$

4. THỬ NGHIỆM VÀ ĐÁNH GIÁ

4.1. Thử nghiệm với SHARPE

Bằng công cụ SHARPE do GS. Trivedi (www.ee.duke.edu/~kst) cung cấp để mô hình hóa hệ thống thực nghiệm [10] chúng ta sẽ so sánh mô hình hệ thống dự phòng nóng áp dụng bảo trì phòng ngừa với mô hình Huang [1, 3] sửa đổi và mô hình hệ thống dự phòng nóng không áp dụng bảo trì phòng ngừa [4, 6]. Từ đó nhằm xem xét ảnh hưởng của sự khác biệt trong thiết kế và sự khác biệt trong môi trường thực thi lên độ tin dùng của hệ thống phòng ngừa nóng.

Để đơn giản hóa trong thử nghiệm chúng ta giả sử rằng $\lambda_3 = \lambda_{31} = \lambda_{32} = \lambda_{33} = \lambda_{34}$. Các tham số thử nghiệm được cho trong bảng sau

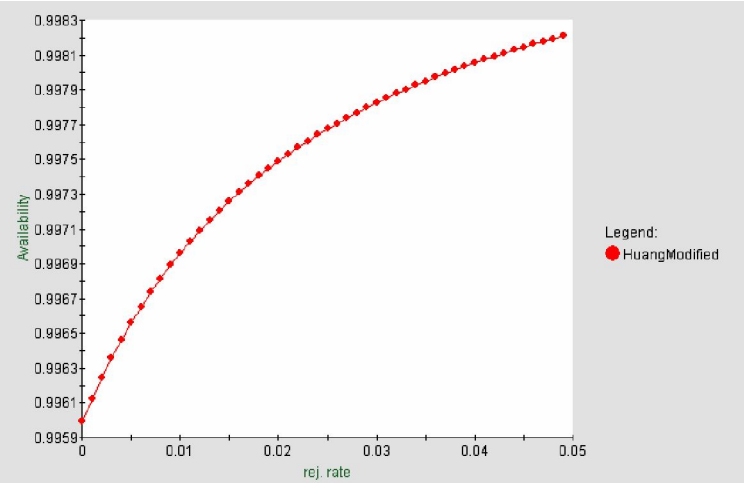
Bảng 2. Các tham số thử nghiệm với SHARPE

Tham số	Giá trị
λ_{11}	240 giờ
λ_{12}	50 giờ
λ_{21}	240 giờ
λ_{22}	50 giờ
r_1	6 /giờ
r_2	6 /giờ
r_3	1 /giờ
r_4	1 /giờ

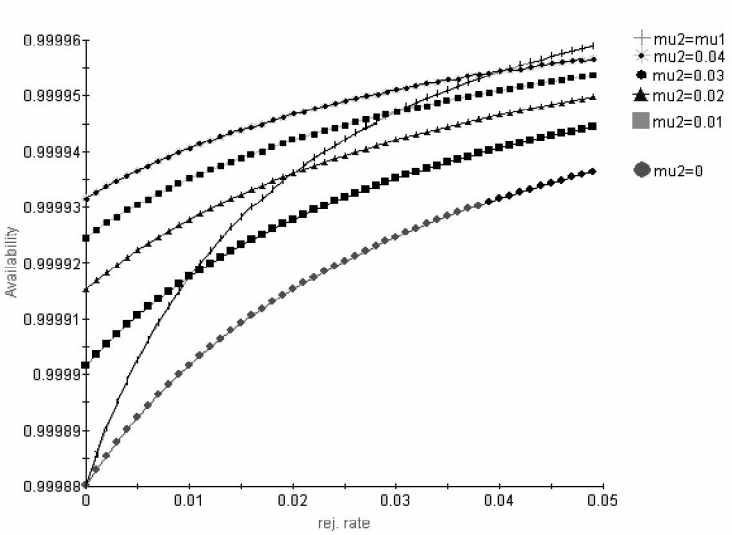
Hình 3 biểu diễn độ sẵn sàng ở trạng thái ổn định trong mô hình của Trivedi. Có thể thấy độ sẵn sàng ở trạng thái ổn định của hệ thống có thể được cải thiện từ $0,997$ đến $0,998(0,1003\%)$ khi thay đổi tốc độ bảo trì phòng ngừa từ $0,001$ sang $0,04$.

Với hệ thống dự phòng nóng không áp dụng bảo trì phòng ngừa, với các tham số ở trên ta có độ sẵn sàng của hệ thống thu được là $0,999909573$.

Độ sẵn sàng của hệ thống dự phòng nóng khi áp dụng bảo trì phòng ngừa được biểu diễn trong Hình 4 và Hình 5 tương ứng với hai trường hợp $\lambda_3 = 1/500$ và$\lambda_3 = 0$. Trường hợp $\lambda_3 = 0$ tương ứng với trường hợp hai phiên bản là hoàn toàn độc lập dưới góc độ lỗi xảy ra trong phiên bản. Ta có thể thấy được với tốc độ bảo trì phòng ngừa $\mu_1 = \mu_2 = 0,05$ hệ thống đạt được độ sẵn sàng là $0,99996$ (so với $0,99985$). So sánh hai đồ thị ta thấy rằng tham số tương quan λ_3 ảnh hưởng mạnh lên độ sẵn sàng của hệ thống, cụ thể là khi λ_3 tăng lên thì sự tương quan giữa hai phiên bản trong việc xảy ra lỗi càng lớn làm cho độ sẵn sàng của hệ thống giảm xuống.



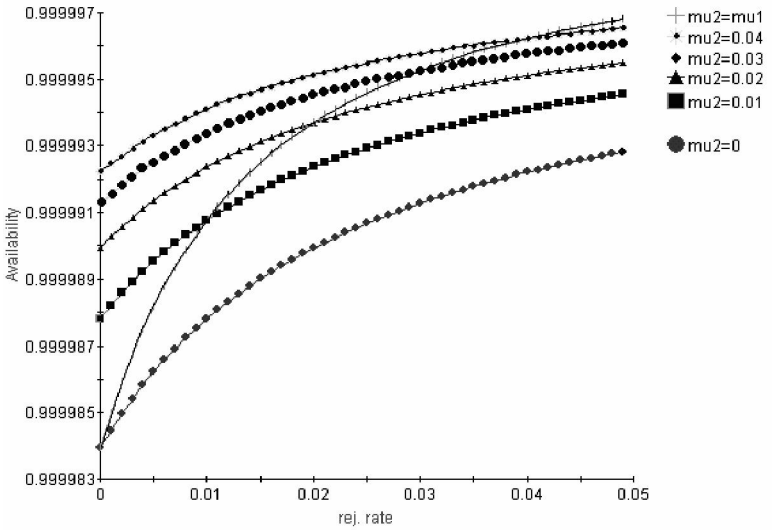
Hình 3. Độ sẵn sàng trong mô hình của Trivedi



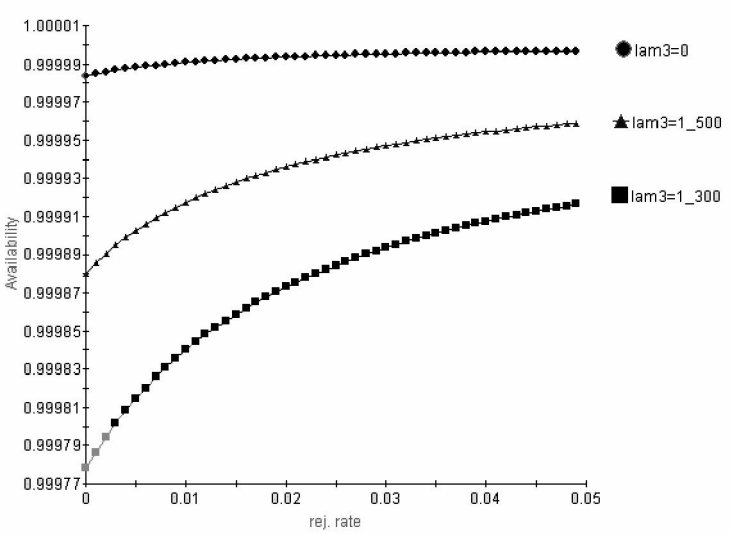
Hình 4. Độ sẵn sàng của hệ thống dự phòng nóng với $\lambda_3 = 1/500$

Hình 6 thể hiện ảnh hưởng của hệ số tương quan giữa hai phiên bản đến độ sẵn sàng của hệ thống. Chúng ta có thể thay được độ sắn sàng của hệ thống thay đổi khá nhanh theo λ_3, đạt cực đại khi λ_3, hai hệ thống hoàn toàn độc lập.

Đánh giá chung: Như vậy, có thể kết luận độ sẵn sàng của phần mềm có thể được điều khiển bởi mô hình hai tham số sự đa dạng trong phát triển và sự đa dạng trong môi trường thực thi. Hai phiên bản được phát triển độc lập theo nguyên tắc của phương pháp NVP, hệ số tương quan của hai phiên bản ảnh hưởng mạnh lên độ sẵn sàng của hệ thống. Trong khi đó, tốc độ Rejuvenation có thể được xác định để ngăn ngừa tình trạng lão hóa tiến trình trong hai phiên bản, từ đó nâng cao độ sẵn sàng.



Hình 5. Độ sẵn sàng của hệ dự phòng nóng với $\lambda_3 = 0$



Hình 6. Độ sẵn sàng của hệ dự phòng nóng với các giá trị khác nhau của λ_3

4.2. Thử nghiệm trên thực tế

Quá trình thử nghiệm mô hình trên thực tế được thực hiện tại thư viện Tạ Quang Bửu Đại học Bách khoa Hà Nội, trong giai đoạn 1 vào tháng 5/2009 (3 tuần) và giai đoạn 2 từ tháng 8/2009 đến 11/2009. Hệ thống được thử nghiệm là Hệ thống phục vụ tra cứu tại thư viện Tạ Quang Bửu, được xây dựng gồm hai phiên bản hoạt động theo cơ chế dự phòng nóng, xuất phát từ một đặc tả yêu cầu ban đầu của hệ thống hai phiên bản trên được phát triển độc lập theo quy trình của kỹ thuật lập trình đa phiên bản:

Phiên bản 1 được phát triển dựa trên nền công nghệ .Net của Microsoft, thực thi trên môi trường IIS web server.

Phiên bản 2 được phát triển dựa trên công nghệ Java Servlet của Sun Micro System, thực thi trên môi trường Tomcat web server.

Hai phiên bản trên được cho hoạt động song song, cùng cung cấp dịch vụ cho người dùng thông qua cơ chế cân bằng tải (load-balancing). Hai phiên bản được áp dụng kỹ thuật Rejuvenation nhằm ngăn ngừa hiện tượng lão hóa tiến trình.

Trong giai đoạn (5/2009), thử nghiệm đồng thời cả hai hệ thống. Trong giai đoạn từ tháng 8/2009 chỉ áp dụng hệ thống hoạt động theo cơ chế bảo trì phòng ngừa. Kỹ thuật bảo trì phòng ngừa ở hai phiên bản trên chỉ dừng lại ở việc dừng hoạt động của ứng dụng, thu dọn tài nguyên của web server. Chúng ta sẽ thống kê thời gian hoạt động của hệ thống, thời gian hệ thống bị lỗi, thời gian bảo trì phòng ngừa, từ đó tính toán độ sẵn sàng của hệ thống và so sánh với hệ thống dự phòng nóng có hai phiên bản giống nhau, không áp dụng bảo trì phòng ngừa.

Các kết quả thu được như sau

Với hệ thống chỉ gồm hai phiên bản giống nhau, không áp dụng bảo trì phòng ngừa:

Bảng 3. Độ sẵn sàng của hệ thống dự phòng nóng

Thông số	Giá trị (Giai đoạn 1: tháng 5/2009)
Tổng thời gian hoạt động	72 giờ = $72 \times 60 \times 60 = 259200$ (s)
Tổng số yêu cầu nhận được	32 348 (yêu cầu)
Tổng thời gian cả hai phiên bản không hoạt động	87(s)
Độ sẵn sàng của hệ thống	$(259200 - 87)/259200 = 0,999664$

Với hệ thống áp dụng mô hình kết hợp bảo trì phòng ngừa và lập trình đa phiên bản áp dụng bảo trì phòng ngừa với cùng tốc độ 2740 yêu cầu/1 lần bảo trì phòng ngừa.

Bảng 4. Độ sẵn sàng của hệ thống dự phòng nóng áp dụng mô hình

Thông số	*Giá trị (Giai đoạn 1: tháng 5/2009)*	*Giá trị (Giai đoạn 2: tháng 8-11/2009)*
Tổng thời gian hoạt động	*72 giờ = 72*60*60 = 259200 (s)*	*960 giờ = 960*60*60 = 345600*
Tổng số yêu cầu nhận được	*28 472 (yêu cầu)*	*48 1394 (yêu cầu)*
Tổng thời gian phiên bản .Net bị lỗi	*243 (s)*	*4108 (s)*
Tông thời gian phiên bản Java bị lỗi	*389 (s)*	*6521 (s)*
Tổng thời gian phiên bản .Net bảo trì phòng ngừa	*847 (s)*	*14319 (s)*
Tổng thời gian phiên bản Java bảo trì phòng ngừa	*679 (s)*	*11323 (s)*
Tổng thời gian cả hai phiên bản không hoạt động	*27 (s)*	*359 (s)*
Độ sẵn sàng của hệ thống	*= (259200-27)/259200 = 0.999895833*	*=(3456000-359)/3456000= 0.999896123*

Nhận xét kết quả

- Khi áp dụng mô hình vào hệ thống, độ sẵn sàng của hệ thống được cải thiện.
- Mặc dù thời gian không hoạt động của mỗi phiên bản tăng lên (tổng thời gian bị lỗi và bảo trì phòng ngừa) tuy nhiên thời gian cả hai phiên bản không hoạt động lại giảm xuống, điều này có thể giải thích do áp dụng bảo trì phòng ngừa, xác suất hệ thống bị lỗi giảm xuống, đặc biệt các lỗi do tương quan giữa hai phiên bản.
- Do thời gian thực nghiệm chưa đủ dài, nên độ sẵn sàng của hệ thống chưa hội tụ về giá trị giới hạn.

Tóm lại, qua số liệu thu được ở trên, ta có thể thấy được việc áp dụng mô hình kết hợp giữa bảo trì phòng ngừa và lập trình đa phiên bản vào hệ thống có cải thiện độ sẵn sàng của hệ thống. Trong thời gian tới, các tác giả sẽ tiếp tục thực hiện việc thử nghiệm với thời gian lớn hơn để có được những kết quả chính xác hơn nữa.

5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Trong bài báo này chúng ta đã nghiên cứu về mô hình kết hợp hai kỹ thuật Rejuvenation và NVP trong đó áp dụng $N = 2$, để tăng cường độ sẵn sàng của phần mềm chịu lỗi, cụ thể là các phần mềm hoạt động liên tục theo thời gian, trong đó xảy ra hiện tượng lão hóa tiến trình. Dựa trên phương pháp dùng chuỗi Markov liên tục về thời gian, chúng ta đã đánh

giá được độ sẵn sàng của hệ thống dự phòng nóng hai phiên bản áp dụng mô hình này, ảnh hưởng của hai tham số cơ bản: tương quan giữa hai phiên bản và tốc độ bảo trì phòng ngừa lên độ sẵn sàng của hệ thống. Bài báo đã chứng minh được sự kết hợp giữa sự đa dạng trong phát triển và sự đa dạng trong môi trường thực thi làm tăng độ sẵn sàng của hệ thống với tốc độ bảo trì phòng ngừa hợp lý.

Trong thời gian tới, để hoàn thiện mô hình chúng ta sẽ xét đến trường hợp trong đó thời gian bảo trì phòng ngừa là xác định và mở rộng chuỗi Markov liên tục về thời gian thành chuỗi semi-Markov để mô hình hóa chính xác hơn các bước chuyển trạng thái trong hệ thống [13]. Thêm vào đó, có thể phát triển mô hình theo hướng xem xét đến trạng thái của hệ thống thay đổi theo tải, phát triển thuật toán để xác định tốc độ bảo trì phòng ngừa tối ưu.

TÀI LIỆU THAM KHẢO

[1] Huang.Y,Kintala C.,Nick K., N. Dudley Fulton, Software rejuvenation: analysis, module and applications, *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing* Pasadena, CA, USA, 1995 (381–391).

[2] Artur Andrzejak, Luis Silva, Deterministic Models of Software Aging and Optimal Rejuvenation Schedules. CoreGRID Technical Report, Number TR-0047, November 2, 2006. Institute on System Architecture. URL: http://www.coregrid.net.

[3] K. Vaidyanathan and K. S. Trivedi, A comprehensive model for software rejuvenation, *IEEE Transactions on Dependable and Secure Computing* **2** (2) (April-June 2005).

[4] Y. Bao, X. Sun, and K. S. Trivedi, A workload-based analysis of software aging and rejuvenation, *IEEE Transactions on Reliability* **54** (3) (2005) (541–548).

[5] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, Analysis of software aging in a Web server, *IEEE Transactions on Reliability* **55** (3) (2006) 411–420.

[6] K. S. Trivedi, K. Vaidynathan and K. Goseva-Popstojanova, Analysis of Software Aging and Rejuvenation, in Recent Developments in Quality Reliability and Information Technology-Trends and Future Directions, P. K. Kapur, A. Kumar, Y. Singh, P. C. Jha, and A. K. Bardhanm (editors), IMH, New Delhi, INDIA, 2004 (25–42).

[7] T.Dohi, K. Goseva-Popstojanova, K. Vaidyanathan, K. S. Trivedi and S. Osaki, Software Rejuvenation- Modeling and Applications, in Springer Handbook of Reliability, H.Pham (editor), Springer Verlag, Feb. 2003 (245–263).

[8] Pham Thanh Trung, Huynh Quyet Thang, Building the reliability prediction model of component-based software architectures, *International Journal of Information Technology* **5** (1) (2009) 17–25.

[9] Huynh Quyet Thang, Pham Ba Quang, Nguyen Tien Dat, Investigate the relation between the correctness and the number of versions of fault tolerant software systems, *International Journal of Computer Science and Engineering* **1** (1) (2007) ISSN 1307-3699, 18–23.

[10] Kisho S. Trivedi, Robin Sahner, SHARPE at the age of twenty two, *ACM SIGMETRICS Performance Evaluation Review* **36** (4) (March 2009) SPECIAL ISSUE: Special issue on tools for computer performance modeling and reliability analysis, pages 52–57.

[11] Phạm Bá Quang, Đào Ngọc Kiên, Huỳnh Quyết Thắng, Mô hình phần mềm chịu lỗi BK-FTS: thử nghiệm, so sánh và đánh giá, *Tạp chí Tin học và Điều khiển học* **21** (4) (2005) 323–336.

[12] Igirdas Avizienis, *The Methodology of N-Version Programming*, in R. Lieu, editor, Software Fault Tolerance, John Wiley & Sons, 1995.

[13] W. K. Ching, K. Ng. Michael, *Markov Chains - Models, Algorithms and Applications*, Springer Science+Business Media, Inc., 2006.

Nhận bài ngày 9 - 9 - 2009
Nhận lại sau sửa ngày 5 - 1 -2010