ISSN: 2175-1897

Tecnologia, inovação e sustentabilidade:
50 anos de Cursos de Tecnologia no Brasil

Avaliação de desempenho de *software* utilizando testes de desempenho: Uma experiência de intervenção

Flavio Augusto Silva[1]

Marilia Macorin de Azevedo[2]

Resumo - O teste de desempenho de software é um processo de coleta e análise dos dados obtidos em requisições simuladas de usuários para identificação de gargalos e pontos de melhoria que podem elevar o desempenho da aplicação. O presente artigo se caracteriza como uma pesquisa qualitativa, no formato de um relato técnico, a partir de uma intervenção em um sistema de arquitetura cliente/servidor que apresentava resultados insatisfatórios nos testes de requisitos não funcionais. Após a intervenção, foi possível observar uma evolução no desempenho da aplicação e um aumento do número de usuários, que passou de 250 para 900 usuários simultâneos.

Palavras-chave: Sistemas produtivos; Teste de desempenho; Desenvolvimento de software.

Abstract - Software performance testing is a process of collecting and analyzing data obtained from simulated user requests to identify bottlenecks and improvement points that can increase application performance. This paper is characterized as a qualitative research, in the form of a technical report, based on an intervention in a client / server architecture system that presented unsatisfactory results in non-functional testing. After the intervention, it was possible to observe an evolution in the application performance and an increase in the number of users, from 250 to 900 simultaneous users.

Keywords: Production systems; Performance testing; Software development.

[1] Centro Estadual de Educação Tecnológica Paula Souza–CEETEPS, e-mail: flavio.silva@cpspos.sp.gov.br
[2] Centro Estadual de Educação Tecnológica Paula Souza–CEETEPS, e-mail: marilia.azevedo@fatec.sp.gov.br

1. Introdução

O uso de aplicações está presente em todas as situações cotidianas. As aplicações que adotam a arquitetura cliente/servidor devem estar disponíveis aos usuários de forma contínua e sem erros. Os usuários julgam o mérito da performance de um sistema web baseados no tempo de resposta da aplicação (LI; SHI; LI, 2013). É preciso assegurar por meio de teste de desempenho que o sistema atenda às especificações de disponibilidade, confiabilidade e segurança. O teste de desempenho é um processo de coleta e análise dos dados obtidos das requisições simuladas de usuários ao sistema para identificação de gargalos e pontos que podem ser modificados para melhorar a performance do sistema (ZHU; FU; LI, 2010).

O presente trabalho se propõe a analisar o método de intervenção da performance de um sistema baseado na arquitetura cliente servidor, desenvolvido por uma empresa de grande porte de TI.

Segundo Myalapalli e Geloth (2015) o ajuste de performance deve ser feito na aplicação, servidor, sistema operacional e no Java Runtime system (caso a aplicação seja desenvolvida na linguagem Java). Um exemplo de ajuste no *Java Runtime system* seria otimizar o gerenciamento automático de memória utilizando o algoritmo de *Garbage Collection* mais adequado (Pufek; Grgić; Mihaljević, 2019).

Esse trabalho está organizado da seguinte forma: na fundamentação teórica são apresentados conceitos básicos de engenharia de software, requisitos, requisitos não funcionais e testes de desempenho. No método está descrito como foi realizada a intervenção. Na seção de resultados obtidos são detalhados os dados resultantes da intervenção. Por fim, nas considerações são discutidas as conclusões obtidas durante a condução deste trabalho.

2. Referencial Teórico

Sommerville (2011) afirma que a engenharia de software é um ramo da engenharia que está evolvida em todos os aspectos da produção de software, desde os estágios iniciais da especificação até a manutenção de sistemas que já se encontram em ambiente produtivo. Segundo Sommerville (2011), a engenharia de software utiliza uma abordagem sistemática, também definida como processo de software, que leva à produção de um produto de software.

Existem muitos processos de software distintos, mas todos possuem quatro atividades fundamentais:

a) Especificação de software
b) Projeto e implementação de software
c) Validação de software
d) Evolução de software

A especificação de software trata de aspectos da funcionalidade do software e as restrições definidas. A atividade de projeto e implementação de software converte as especificações em um sistema executável. A validação de software tem a intenção de mostrar que um software se adequa às suas especificações ao mesmo tempo que atende as necessidades dos clientes. A atividade de validação também envolve processos de verificação em cada estágio do processo de software (SOMMERVILLE, 2011).

Segundo Kotonia e Sommerville (1998), requisitos de software são especificações de serviço que um sistema deve prover, restrições e um conjunto de conhecimentos necessários para desenvolver um software. A engenharia de requisitos é um sub-ramo da engenharia de software que aborda o processo de definição dos requisitos de software. Envolve um conjunto de atividades como elicitação, análise e negociação, documentação, validação e gestão de requisitos (KOTONIA; SOMMERVILLE, 1998).

Os requisitos podem ser classificados em funcionais ou não funcionais. Os requisitos funcionais são declarações de serviço que o sistema deve fornecer e o comportamento determinado que o sistema deve apresentar devido a entradas de dados específicas (SOMMERVILLE, 2011). Os requisitos não funcionais (RNF) cuidam de aspectos gerais de requisito do sistema como custo, performance, confiabilidade, manutenabilidade, portabilidade e custos operacionais (CHUNG et al., 1999).

Testes de desempenho podem ser realizados para verificar se o sistema atende aos RNF especificados. O teste de desempenho mede e avalia o grau em que um item de teste desempenha suas funções designadas dentro de determinados limites de tempo, uso de recurso (CPU, disco, memória, rede) e de capacidade. O teste de capacidade é um tipo de teste de desempenho que serve para avaliar o nível no qual o aumento de carga (de usuários, armazenamento de dados e transações) compromete a capacidade de um item de teste para sustentar o desempenho requerido (ISO/IEC/IEEE, 2013).

Garbage Collection é uma ferramenta utilizada para o gerenciamento da memória de programas, que automaticamente libera a memória que não é mais utilizada. A utilização de *Garbage Collection* facilita o trabalho do programador, pois ele não precisa se preocupar com aspectos de uso de memória pelo programa, porém tem o inconveniente de concorrer com o programa por recursos da CPU. Van Hoff (1997) afirma que o algoritmo de implementação do *Garbage Collection* influencia no tempo de parada de processamento da aplicação; o algoritmo geracional, por exemplo, reduz a parada de processamento de 100 a 200 milissegundos para 1 a 2 milissegundos.

3. Método

O relato técnico foi feito com base na experiência profissional do pesquisador e análise documental referente à intervenção ocorrida em 2017 em uma aplicação para resolução de problema de suportar baixo número de

usuários simultâneos observado em testes de requisitos funcionais. A intervenção foi feita em uma empresa que atua no segmento de Tecnologia da Informação com cerca de 9000 funcionários e faturamento anual aproximado de três bilhões de reais.

A aplicação utiliza o modelo cliente servidor, foi desenvolvida em linguagem Java e utiliza banco de dados Oracle e servidor JBoss AS7. Está dividida em três camadas (arquitetura *Model View Controller*), sendo que a camada de visão foi desenvolvida utilizando a tecnologia Java Server Faces (JSF).

O desenvolvimento do trabalho foi dividido em três fases:

1) Avaliar a quantidade limite de usuários suportada pela aplicação
2) Identificar melhorias na aplicação para diminuir o consumo de memória e tempo de resposta das requisições
3) Reavaliar a quantidade limite de usuários suportada pela aplicação após implementação de melhorias

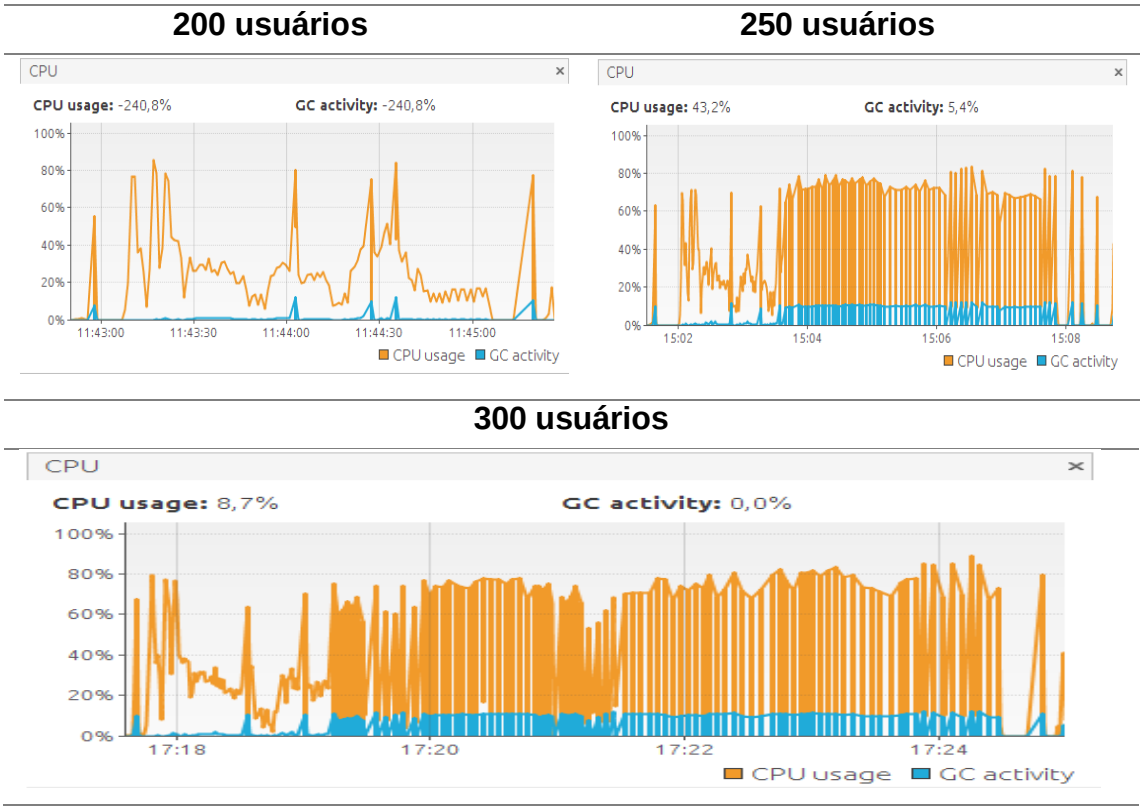
Para realizar as tarefas propostas foram feitos testes de desempenho em ambiente de RNF, com configurações equivalentes ao ambiente produtivo. Foram escolhidas três funcionalidades críticas para a realização dos testes não funcionais. As ferramentas utilizadas para a solução do problema foram:

a) *Apache Jmeter* – utilizado para executar os testes de RNF. A ferramenta permite planejar e executar os testes de RNF, bem como disponibilizar relatórios para análise de desempenho a partir dos dados de testes coletados.
b) *Eclipse Oxygen* – utilizado para recuperar o código fonte do repositório, fazer modificações nesses e empacotá-los para utilização no servidor de aplicação.
c) *JBoss* AS7 – servidor de aplicação para executar o sistema e submeter a aplicação aos testes de carga.
d) *Oracle Java VisualVM* – ferramenta utilizada para acompanhar a utilização de recursos (uso de memória e CPU), comportamento do processo Java como, por exemplo, *threads*, *garbage collection* e *permgen*.

4. Resultados e Discussão

A primeira fase do trabalho foi avaliar a quantidade limite de usuários da aplicação e chegou-se ao total de 200 usuários simultâneos. Ao observar a figura 1 para a bateria de 250 usuários, foi possível identificar que aplicação começou a apresentar degradação porque o *Garbage Collection* estava concorrendo com a aplicação a partir de determinado momento.

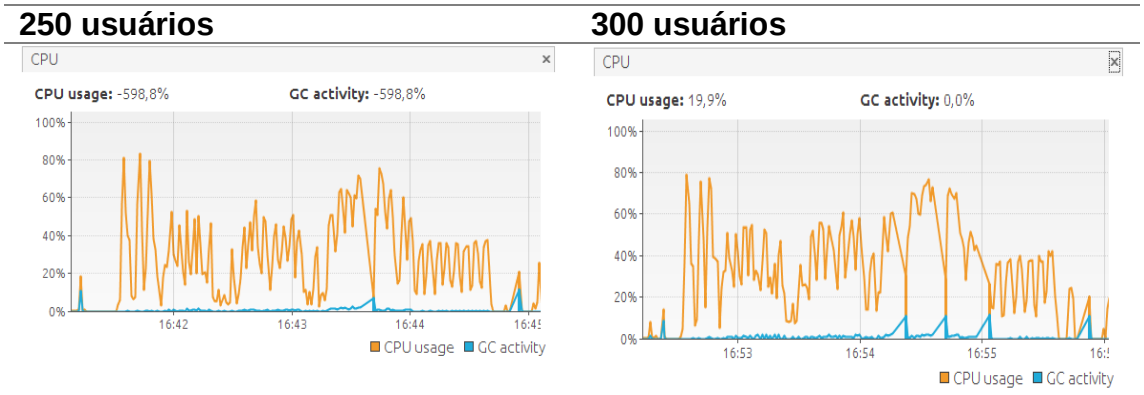
Figura 1- Uso de CPU+GC – Aplicação e Servidor padrão

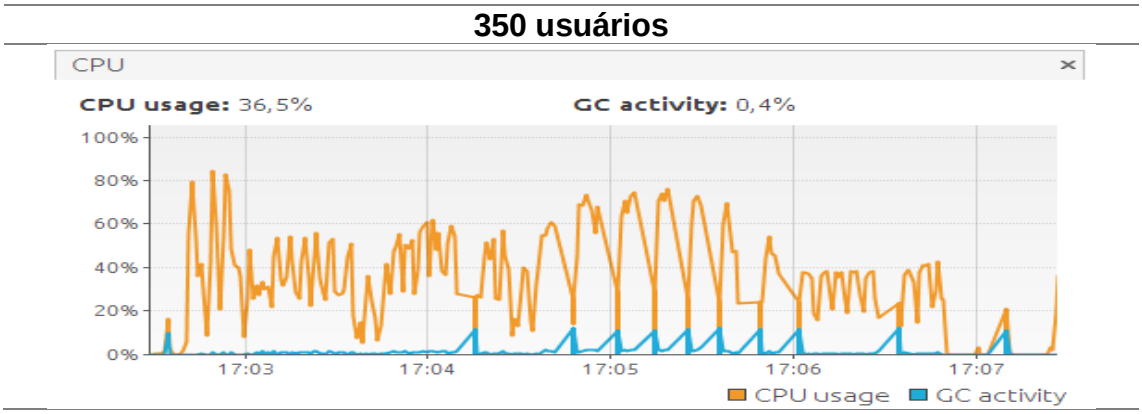


Fonte: Relatório de análise de resultados dos testes de RNF

A alta atividade do *Garbage Collection (GC)* indicou que o algoritmo de GC utilizado pelo servidor deveria ser alterado para um mais performático. Assim, o servidor foi configurado para utilizar o algoritmo G1CG para atender essa necessidade. Uma nova bateria de testes foi realizada e foi possível observar que o limite de usuários simultâneos aumentou para 350 usuários sem apresentar erros, conforme demonstrado na figura 2.

Figura 2 - Uso de CPU+GC – Aplicação e Servidor com G1GC

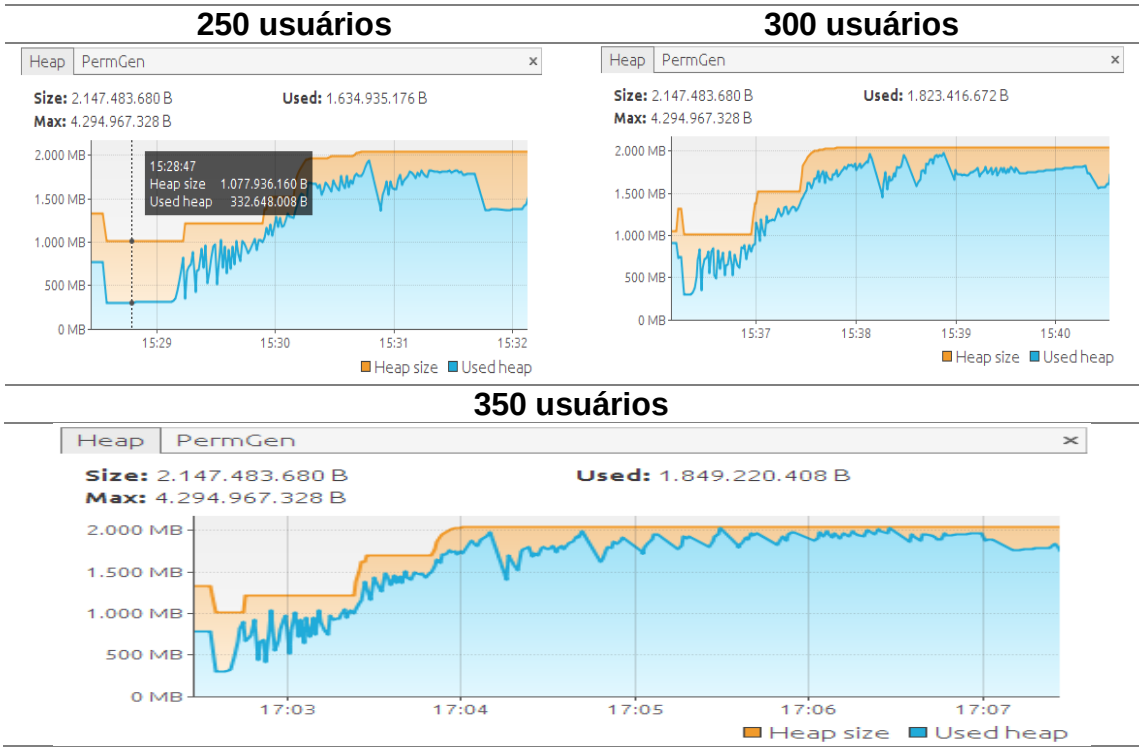




Fonte: Relatório de análise de resultados dos testes de RNF

A partir de 400 usuários simultâneos a aplicação passou a registrar erros na execução devido à pouca memória disponível. Na figura 3 é possível observar que com 350 usuários a memória *Heap* disponível foi quase toda utilizada.

Figura 3 - Uso de Memória Heap – Aplicação e Servidor com G1GC



Fonte: Relatório de análise de resultados dos testes de RNF

A segunda fase do trabalho foi identificar formas de melhorar o uso de memória e diminuir o tempo de resposta da aplicação. Uma bateria de testes com 400 usuários simultâneos foi utilizada para se obter o tempo de resposta de cada funcionalidade, apresentado na tabela 1.

Tabela 1 - Tempo de resposta por funcionalidade

Funcionalidade	Tempo de resposta
Acesso Sistema	1,428 seg
Acesso Consulta Processo	1,677 seg
Execução Consulta Processo	1,745 seg

Fonte: Relatório de análise de resultados dos testes de RNF

De acordo com a tabela 1, os tempos de resposta obtidos foram considerados altos. Dessa forma, foi registrado o tempo de resposta dos métodos de negócio de cada funcionalidade para avaliar a causa da demora no tempo de resposta. Com a análise foi possível observar que a maior parte do tempo de resposta era despendido com as funções de acesso aos dados. Para cada uma das funcionalidades testadas foi avaliado como a aplicação se comportava em relação às atividades de acesso de dados.

4.1 Problemas na funcionalidade Acesso ao sistema

Na funcionalidade de acesso ao sistema observou-se que as consultas de tabelas de apoio sempre eram executadas em todos os testes (Problema 1). Ao executar a aplicação utilizando o *browser* para verificar a apresentação das tabelas de apoio, foi possível verificar que as mesmas eram carregadas na inicialização de um *ManageBean* e eram disponibilizadas em um modal que não era sempre exibido. Esse trecho de código fazia parte de um *template*; assim, o problema era propagado por toda aplicação. A solução aplicada foi carregar as tabelas de apoio somente em situações específicas. Os tempos obtidos são apresentados na tabela 2.

Tabela 2 - Tempos médios por funcionalidade - 400 usuários – Problema 1

Antes

Label	# Samples	Average
WarmUp - Ace...	20	9061
WarmUp - Exi...	20	3953
WarmUp - Exe...	20	4559
Acessando o ...	400	53158
Acessando a ...	400	100628
Executando C...	400	85690
TOTAL	1260	76303

Depois

Label	# Samples	Average
WarmUp - Ace...	20	5311
WarmUp - Exi...	20	5021
WarmUp - Exe...	20	6782
Acessando o ...	400	10443
Acessando a ...	400	74489
Executando C...	400	125371
TOTAL	1260	67035

Fonte: Relatório de análise de resultados dos testes de RNF

A tabela 2 mostra que houve uma melhora no tempo médio de resposta das funcionalidades.

4.2 Problemas na funcionalidade Acesso a consulta de processos

Na funcionalidade de acesso a consulta de processos foi observado um problema semelhante ao que ocorria na funcionalidade de acesso ao sistema. Haviam tabelas de apoio que eram carregadas e exibidas em um modal que não era exibido em todas as situações (Problema 2). A mesma solução dada anteriormente foi aplicada. Uma nova bateria de testes foi aplicada para verificar a situação da aplicação, apresentada na tabela 3.

Tabela 3 - Tempo médio de resposta por funcionalidade – 400 usuários - Problema 2

Antes

Label	# Samples	Average
WarmUp - Ace...	20	5311
WarmUp - Exi...	20	5021
WarmUp - Exe...	20	6782
Acessando o ...	400	10443
Acessando a ...	400	74489
Executando C...	400	125371
TOTAL	1260	67035

Depois

Label	# Samples	Average
WarmUp - Ac...	20	6073
WarmUp - Exi...	20	1499
WarmUp - Ex...	20	5704
Acessando o ...	400	7688
Acessando a ...	400	8761
Executando ...	400	51062
TOTAL	1260	21643

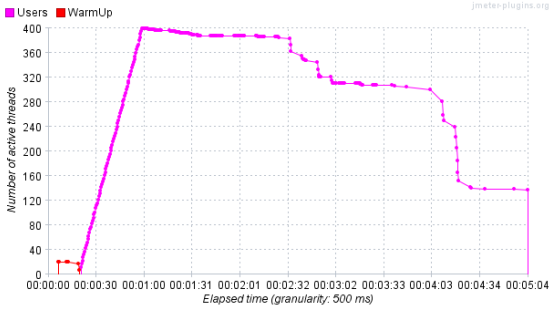
Fonte: Relatório de análise de resultados dos testes de RNF

A tabela 3 mostra que houve uma melhora significativa em todas as consultas, não apenas na consulta de processos. Essa melhora pode ser explicada por haver menos concorrência entre funcionalidades em paralelo.

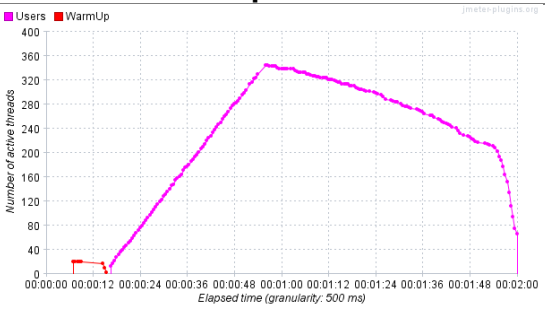
Observou-se também o comportamento do ambiente com 400 usuários simultâneos, conforme figura 4.

Figura 4 - Usuários ativos no tempo - 400 usuários – Problema 2

Antes



Depois



Fonte: Relatório de análise de resultados dos testes de RNF

A figura 4 mostra que houve uma diminuição na quantidade de *usuários ativos* no tempo e que o teste foi concluído em menos da metade do tempo.

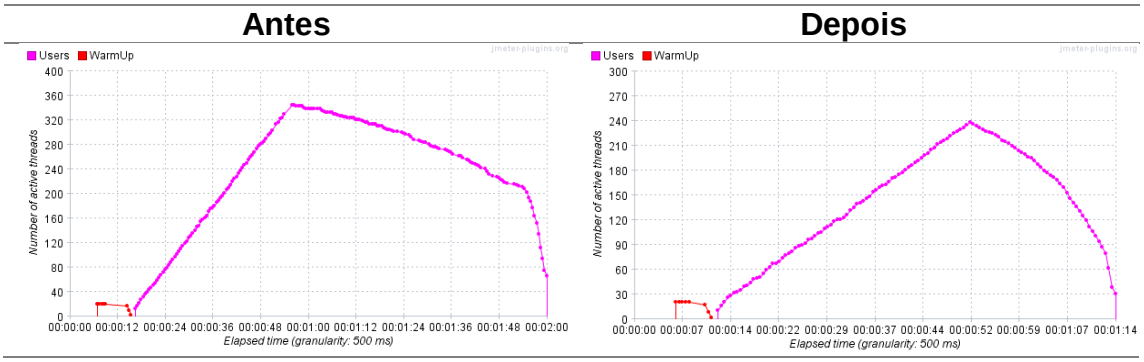
4.3 Problemas na funcionalidade Execução Consulta Processo

Para essa funcionalidade constatou-se três problemas principais:

a) Várias entidades eram consultadas, porém apenas alguns dados eram realmente relevantes para a aplicação.
b) Alguns dados necessários para aplicação eram carregados tardiamente (*Lazy*) quando já poderiam ser retornados na consulta original.
c) Algumas entidades não necessitavam de todos os dados relacionados às associações; assim, foram alterados para incluir o carregamento tardio em suas associações.

As devidas correções no código foram efetuadas e uma nova bateria de testes foi realizada, e pode-se observar o comportamento na figura 5.

Figura 5 - Usuários ativos no tempo - 400 usuários



Fonte: Relatório de análise de resultados dos testes de RNF

A figura 5 mostra que houve uma diminuição de 48s no tempo total de execução de testes, bem como a diminuição do número de usuários ativos que passou de 360 para cerca de 240 usuários ativos

A última etapa do trabalho foi reavaliar a quantidade máxima de usuários ativos para a aplicação. Partindo de 400 usuários, as baterias de testes foram sendo incrementadas em 100 usuários até encontrar um novo limite no qual a aplicação começa a apresentar erros. Nessa etapa foi constatado que a aplicação começou a apresentar erros com 900 usuários simultâneos.

5. Considerações finais

Após a intervenção na aplicação o número máximo de usuários simultâneos aumentou de 250 para 900 usuários. Comparando as figuras 4 e 5 é possível observar que houve uma redução de 250 segundos na execução da bateria de testes, evidenciando a melhora do desempenho da aplicação.

Foi possível aumentar a capacidade de atendimento a usuários simultâneos da aplicação com a mudança de estratégia de *Garbage Collection* e poucas alterações no código. Sugere-se avaliar outras funcionalidades da aplicação para verificar se existem problemas de desempenho.

Sugere-se a divulgação das práticas adotadas nesse processo de intervenção a outras equipes para se evitar que esses tipos de problemas ocorram em outros projetos de sistemas desenvolvidos dentro da empresa.

Outra sugestão é a disseminação de treinamentos para o corpo funcional sobre técnicas avançadas sobre a utilização do mapeamento de entidades na linguagem Java (mapeamento *Hibernate*/*JPA*).

Referências

CHUNG, L.; NIXON, B.; YU, E.; MYLOPOULOS, J. *Non-Functional Requirements* in Software Engineering: Kluwer Academic Publishers, 1999.

ISO/IEC/IEEE. *Software and systems engineering - Software testing - Part* 1: *Concepts and definitions*. *Standard* 29119-1:2013 (E), *International Organization for Standardization* (ISO)/*International Electrotechnical Commission* (IEC*)/Institute of Electrical and Electronics Engineers* (IEEE). p. 1-64. 2013.

KOTONYA, G.; SOMMERVILLE, I. *Requirements Engineering: Processes and Techniques. Chichester*, UK: John Willey & Sons, 1998.

LI, P.; SHI, D.; LI, J.; *Performance test and bottle analysis based on scientific research management platform: 10th International Computer Conference on Wavelet Active Media Technology and Information Processing* (ICCWAMTIP): IEEE, 2013.

MYALAPALLI, V. K.; GELOTH, S*. High Performance JAVA Programming. International Conference on Pervasive Computing* (ICPC): IEEE, 2015

PUFEK, P.; GRGIĆ, H.; MIHALJEVIĆ, B. Analysis of Garbage Collection Algorithms and Memory Management in Java. 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO): IEEE, 2019.

SOMMERVILLE, I. Engenharia de Software. 9a edição. São Paulo: Pearson Prentice Hall, 2011.

VAN HOFF, A. *The case for Java as a programming language*: IEEE Internet Computing, 1997. 51-56 p.

ZHU, K.; FU, J.; LI, Y. *Research the performance testing and performance improvement strategy in web application*. *2nd International Conference on Education Technology and Computer*: IEEE, 2010.