Desempenho de Aplicações Web: Um estudo comparativo utilizando o software Redis

Pedro Jannotti Sampaio, Igor de Oliveira Knop

Centro de Ensino Superior de Juiz de Fora (CES/JF) - Juiz de Fora, MG - Brasil

pedrojnt@gmail.com

Abstract. *One of the main points to web systems users is the fast availability of the content. But with the huge volumn of data that are stored, the growing amount of users and the need to supply real-time content become dificult supply the content with a acceptable performace. In this article will be presented the concepts of data cache and some approachs to reduce the response time of an application with high number of requisitions. It is presented a case study where the requisitions processing time of two applications are compared, one withou cache and another with cache via Redis software. The application with use of cache presents a lower response time, showing a performance improvement and justifying the use of cache on the applications with the same nature.*

Resumo: *Um dos pontos principais para melhorar a experiência dos usuários de sistemas web é a rápida disponibilidade do conteúdo. Porém com o grande volume de dados armazenados, a crescente quantidade de usuários e a necessidade de conteúdo em tempo real se torna difícil fornecer o conteúdo com um desempenho aceitável. Neste artigo será apresentado os conceitos de cache de dados e algumas abordagens para diminuir o tempo de resposta de uma aplicação com alto número de requisições. É apresentado um estudo de caso onde se compara os tempos de processamento das requisições de duas aplicações, uma sem uso de cache e a outra com uso de cache via software Redis. A aplicação com o uso de cache apresenta um tempo bem menor de resposta, evidenciando um ganho de desempenho e justificado a utilização de cache em aplicações de mesma natureza.*

1. Introdução

A Internet continua em rápido crescimento do número de usuários, servidores web e dados trafegados. A forma como as pessoas usam a Internet mudou devido a adoção de novas tecnologias, o comportamento da indústria com o aumento do acesso a Internet, entre outros fatores. A média de dados transferidos por usuários por mês passou de 12 *gigabytes* para 22 *gigabytes* em apenas um ano [Gebert *et al.* 2012] mostrando o salto de conectividade da população mundial e a importância que a Internet tomou na vida das pessoas.

Com esse aumento de utilização também foi observado um aumento de carga nas redes e no tempo de resposta das aplicações ao usuário [Pitkow e Recker *apud* Viles e French 1994]. O problema não é somente de infraestrutura da rede mas também como as aplicações são construídas.

Muitas abordagens foram propostas para minimizar o problema: melhorias no desenvolvimento de aplicativos, servir conteúdo estático e dinâmico de forma mais

eficiente [Nygren, Sitaraman e Sun 2010], compressão de cabeçalhos e requisições [Drew 2008], melhorias na forma de roteamento do tráfego pela rede através do uso do protocolo OSPF (*Open Shortest Path First*) [Moy 1998]. Entretanto, um dos métodos mais comuns para se aproveitar melhor a rede é o cache.

Cache, em sua definição, é uma técnica que consiste em armazenar um conjunto de dados temporariamente em um ambiente que permita um acesso rápido do processador, diferente do armazenamento convencional (em disco) [Evans 2014]. Cache pode ser usado em diversos níveis na web, como a resolução de nomes através de cache de DNS, que é o cache de endereços IP relacionados ao domínio do site acessado. Cache de blocos e páginas web através de ferramentas específicas de Web Proxy Caching. Cache do HTTP, o protocolo de transferência utilizado atualmente na Internet permite o uso de cache de páginas e arquivos estáticos.

Nesse artigo será abordado o uso de cache de banco de dados, onde o objetivo é otimizar o acesso a dados e a escrita de dados que são utilizados constantemente.

2. Cache de Banco de Dados

A grande maioria das aplicações web possuem uma forma de armazenamento de dados. O armazenamento é extremamente importante no processo de negócios para aplicações. O armazenamento de dados pode ser usado para monitorar o comportamento de usuários para exibição de anúncios, criação de conteúdo, marketing, sugestão de conteúdo, desenvolvimento de pesquisas científicas, como mapeamentos biológicos e físicos, melhoria de plataformas voltadas para interação com o usuário como plataformas educacionais, jogos, entre outros.

Sistemas de gerenciamento de dados relacionais são os mais populares para o armazenamento de dados. A lista da Figura 1 apresenta dados compilados pela empresa Solid IT que cataloga a popularidade de banco de dados através de dados de busca, fóruns de discussão, quantidade de empregos oferecidos e relevância através de redes sociais [DB-Engines 2015].

283 systems in ranking, October 2015

Rank			DBMS	Database Model	Score		
Oct 2015	Sep 2015	Oct 2014			Oct 2015	Sep 2015	Oct 2014
1.	1.	1.	Oracle	Relational DBMS	1466.95	+3.58	-4.95
2.	2.	2.	MySQL	Relational DBMS	1278.96	+1.21	+15.99
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1123.23	+25.40	-96.37
4.	4.	↑ 5.	MongoDB +	Document store	293.27	-7.30	+52.86
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	282.13	-4.05	+24.41
6.	6.	6.	DB2	Relational DBMS	206.81	-2.33	-0.86
7.	7.	7.	Microsoft Access	Relational DBMS	141.83	-4.17	+0.19
8.	8.	↑ 10.	Cassandra +	Wide column store	129.01	+1.41	+43.30
9.	9.	↓ 8.	SQLite	Relational DBMS	102.67	-4.99	+7.71
10.	10.	↑ 12.	Redis +	Key-value store	98.80	-1.86	+19.42

Figura 1. Ranking de popularidade de bancos de dados [Solid IT 2015]

Os bancos de dados relacionais presentes na lista da Figura 1 utilizam o armazenamento de dados em disco rígido. O grande problema desta abordagem é que o acesso ao disco é cara do ponto de vista computacional, o que pode prejudicar o desempenho da aplicação quando é estressada, pois o tempo de acesso ao disco rígido é

maior, em grande magnitude, do que o acesso a memória principal [Garcia-Molina e Salem 1992]. Outra abordagem utilizando armazenamento em disco é a utilização de discos de estado sólido ao invés de disco rígido. Esta tecnologia permite rápido acesso e gravação de dados, porém o custo de aquisição do armazenamento é muito superior ao do disco rígido, sendo necessário a análise das melhores ocasiões de utilização da tecnologia [Dell 2011].

Sistemas de cache entram nesse ponto para aumentar a velocidade de recuperação e escrita dos dados armazenados a um baixo custo.

Normalmente o tamanho do cache é reduzido, já que memórias de acesso rápido geralmente são mais caras do que o armazenamento comum. Desta forma não é possível armazenar todos os dados do banco de dados no próprio cache, sendo necessário a existência de políticas de substituíção para otimizar a utilização do cache [Lorenzetti, Rizzo e Vicisano 2000].

No mercado existem diversas soluções para utilização de cache no desenvolvimento, tais como: AppFabric Caching, um serviço que aumenta as capacidades do Windows Server fornecendo uma API para gerenciamento de cache [MSDN 2011]; Memcached, um sistema de armazenamento em chave-valor que é utilizado através de bibliotecas auxiliares às linguagens de programação [Memcached 2015] e diversas outras soluções que não foram desenvolvidas para cache, mas podem ser usadas para esse fim, como: Redis, um sistema de armazenamento em memória de chave-valor [Redis 2015]; Riak, um serviço de armazenamento de objetos que pode ser utilizado em diversas linguagens de programação [Basho 2015]; Dynamo DB, um sistema de armazenamento de dados proprietário que pode ser utilizado através da API do próprio fabricante [Amazon 2015] entre outros.

Alguns dos bancos de dados mais utilizados no mercado também possuem suas próprias soluções internas para tratar o cache de dados, podendo ser utilizado sem a necessidade de instalação de ferramentas de terceiros. A ferramenta MySQL apresenta cache de *queries* e *buffer* para os diferentes tipos de armazenamentos que utiliza [MySQL Cache 2015], assim como o banco de dados da Oracle apresenta o cache de resultados para melhorar o desempenho da recuperação de dados em seu armazenamento [Nanda 2015].

2.1. Estratégias de Cache

Ao utilizar cache para melhorar o desempenho é preciso determinar as melhores estratégias para manipulação, de acordo com o objetivo da aplicação. No caso desse estudo de caso, é preciso definir as estratégias de inserção de blocos do cache e a alteração de dados dentro do cache.

2.1.1. Estratégia de Inserção

Duas ações devem ser consideradas ao executar escritas no cache e políticas específicas para cada situação devem ser elaboradas. A estratégia de inserção de cache deve considerar as ações de *write-miss* e *write-hit*, ou a ação realizada quando um dado não está presente no cache e a ação realizada quando um dado está presente no cache, respectivamente.

As estratégias de *write-miss*, quando o dado não está presente no cache, incluem três variáveis que se intercalam: *write-allocate*, *fetch-on-write* e *write-*

invalidate. Em *write-allocate*, quando é solicitado uma escrita no cache em que um dado não é previamente encontrado, um espaço em cache é alocado para receber o novo dado através do *write-hit*. Em *fetch-on-write*, deve-se primeiro recuperar o conteúdo da chave de cache a ser gravada de outros caches ou mesmo da memória principal, para depois atualizar o cache com *write-hit*. Em *write-invalidate*, todas as cópias de uma determinada chave de cache que deve ser alterada são invalidadas antes da alteração no cache que sofrerá o *write-hit* [Jouppi 1991].

Na ação de *write-hit*, quando o dado já está contido no cache e não foi invalidado, as estratégias utilizadas são *write-through*, *write-around* e *write-back*.

A primeira estratégia é a *write-through cache*, ou escrita através do cache. Essa estratégia consiste em inserir o bloco de informação no cache e também diretamente na memória principal, como pode ser observado na Figura 2.

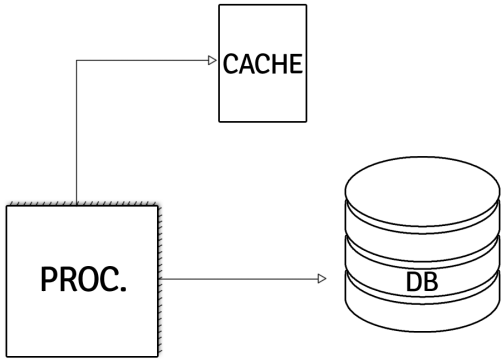

Figura 2. Fluxo da estratégia de *write-through*. Fonte: elaborado pelo autor.

A vantagem de se utilizar essa estratégia é a garantia de que o dado estará armazenado tanto na memória principal quanto no cache. Porém a desvantagem dessa abordagem é que o sistema irá enfrentar os efeitos da latência maior do acesso a memória principal [Evans 2014], em nosso caso o banco de dados.

A segunda estratégia é a *write-around cache*. Nessa estratégia, o dado é salvo diretamente em memória principal, desconsiderando o cache no momento da escrita, como pode ser observado na Figura 3.

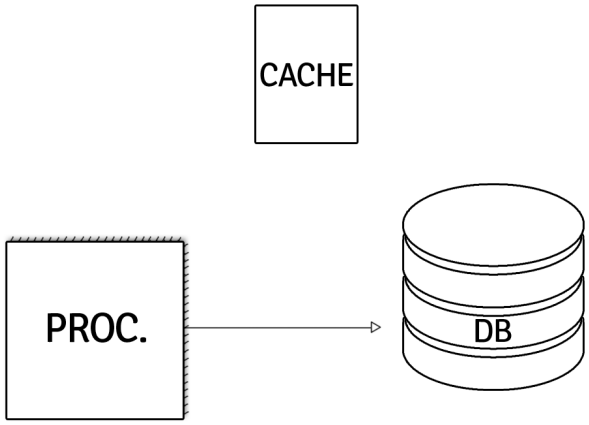

Figura 3. Fluxo da estratégia de *write-around*. Fonte: elaborado pelo autor.

Esta estratégia reduz a escrita excessiva de dados no cache que não será acessada em um futuro próximo. A desvantagem desta abordagem acontece quando um

dado é solicitado para leitura e, como não foi salvo em cache, deverá ser lido de uma fonte mais lenta [Evans 2014] como o banco de dados em nosso caso.

A terceira estratégia é a *write-back cache*. Nessa estratégia, os dados são escritos diretamente, e somente, no cache. Quando o dado no cache é alterado por outro valor, o valor é retirado do cache e armazenado em memória principal, como pode ser observado na Figura 4.

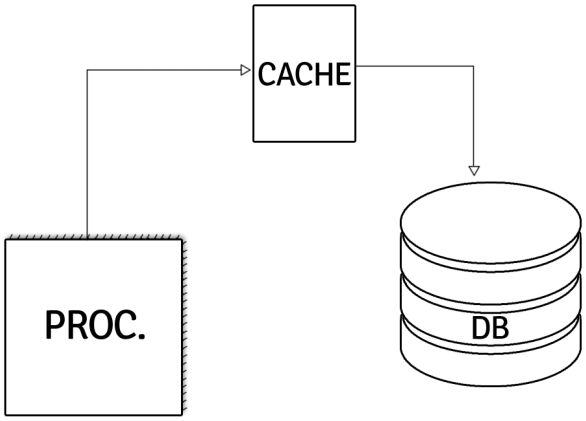

Figura 4. Fluxo da estratégia de *write-back*. Fonte: elaborado pelo autor.

Esta estratégia tem a vantagem de permitir baixa latência e custo através do pouco armazenamento em memória principal para aplicações com grande intensidade de escritas [Evans 2014].

A desvantagem desta abordagem é o risco de perda de dados, pois a única cópia do dado está apenas disponível em cache. Para minimizar esta situação, ferramentas de cache tem desenvolvido formas de redundância de dados através de escritas duplicadas [Evans 2014].

Nesse estudo de caso, o objetivo do uso do cache é melhorar o desempenho do processamento e armazenamento das mensagens enviadas pelos usuários. Nesse cenário a estratégia de *write-back* possui vantagem sobre o *write-through* devido a redução do tráfego direto e frequente à memória principal [Jouppi 1991].

2.1.2. Estratégia de Alteração de dados

Estratégias de alteração de dados são utilizadas quando a capacidade do cache de reter conteúdo é exaurida. O objetivo de se utilizar um algoritmo para a alteração dos dados em cache é reduzir o *cache miss*, ou a quantidade de vezes que é necessário acessar a memória principal para encontrar o dado necessário para o processamento [Arpaci-Dusseau 2015].

Existem no mercado diversos algoritmos com o objetivo de minimizar o *cache miss*, alguns mais simples e outros mais complicados de ser implementado, o que deve ser considerado na escolha de acordo com a aplicação. Dentre as estratégias utilizadas no mercado exitem a FIFO, a implementação de uma fila onde os primeiros registros inseridos serão removidos na necessidade de mais espaço no cache [Arpaci-Dusseau 2015]; Random, através da retirada de registros do cache de forma aleatória [Arpaci-Dusseau 2015]; LFU, que verifica dados de frequência de utilização do registro para remover os que são menos utilizados [Arpaci-Dusseau 2015].

Dentre as estratégias existentes para alteração de dados em cache, será utilizado neste estudo de caso o LRU ou *Least Recently Used*.

LRU é baseado na observação que os dados que são utilizados recentemente serão, provavelmente, usados no futuro. Enquanto, dados que não foram utilizados recentemente, permanecerão não utilizados por um longo tempo. No LRU quando o cache estiver completo, o objeto que não foi utilizado pelo maior tempo é jogado fora.

O cache consiste em uma lista em que os itens mais recentemente referenciados estão em seu início. Quando um novo registro é adicionado, ele é alocado no início da lista e o registro no final é removido. LRU não considera o tamanho do registro em cache [Joy e Jacob 2012].

Uma representação visual do processo pode ser observado abaixo na Figura 5.

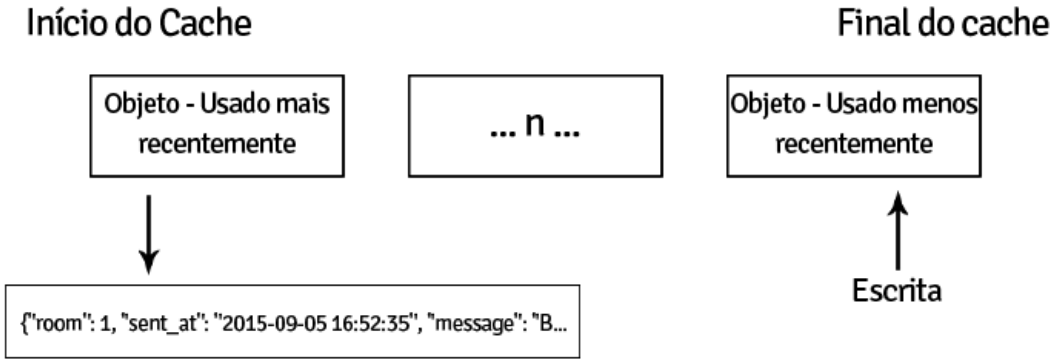

Figura 5. Representação do cache utilizando LRU. Fonte: elaborado pelo autor.

2.2. Redis

O Redis é um sistema de armazenamento e cache baseado em chave-valor avançado e de código livre sob a licença BSD. Sistemas de chave e valor podem ser divididos em dois grupos: armazenamento em memória e em disco. O armazenamento em memória possui alto desempenho, normalmente sendo usado para cache distribuído, enquanto armazenamento em disco são usados como base de dados [Heise 2010].

O nome do software é um acrônimo de *REmote Dictionary Service*, foi liberado para uso em 2009 por Salvatore Sanfilippo. E segundo o próprio, ele foi desenvolvido pela necessidade.

Salvatore Sanfilippo trabalhava em um sistema de análise de dados em tempo real que necessitava realizar operações rápidas de escrita e leitura. O grande problema era que nenhum sistema de banco de dados mantinha um desempenho decente com o volume de informação que ele utilizava.

É uma ferramenta híbrida, a maior parte do seu processamento ocorre em memória, o que garante desempenho, porém também é possível utilizar persistência e armazenamento de dados em disco para manutenção dos dados. Um dos motivos que fizeram a escolha da ferramenta para uso em cache.

Suas estruturas de dados resolvem grande parte dos problemas de armazenamento sem a necessidade de mais implementações como outros bancos de dados [Carlson 2013]. Redis possui suporte nativo, com comandos específicos, para manutenção de listas, mapas, listas de valores únicos, geolocalização, transações,

publicação e subscrição, além dos comandos básicos para manipular strings.

Através de comandos específico, Redis permite o armazenamento de dados em disco, minimizando o risco de perda de dados. Esse armazenamento pode ser realizado automaticamente através de diretivas de configuração da ferramenta que executam o armazenamento de acordo com a quantidade de escritas realizadas no banco de dados do Redis.

Outros motivos para a escolha do Redis é sua vasta documentação e confiabilidade, sendo utilizado extensivamente por grandes empresas do mercado de software como Twitter [Krikorian 2012], Instagram [Instagram 2012] e Github [Github 2009] que armazenam grandes volumes de dados.

3. Estudo de caso

Nesse estudo de caso, foi verificado o comportamento de uma aplicação web antes e depois da aplicação de cache de dados. A aplicação desenvolvida é um sistema de comunicação entre dois ou mais usuários. A comunicação é executada em tempo real e todas as mensagens trocadas pelos usuários são armazenadas para futuras consultas e ou outros tipos de processamentos.

3.1. Arquitetura da aplicação do estudo de caso

A aplicação foi desenvolvida utilizando a linguagem de programação PHP 5.5, através do framework Symfony 2.7. Para o servidor web foi utilizado o Apache 2 e toda a comunicação em tempo real que permite a interação entre os usuários é feita através de Websockets através da especificação do HTML 5. A aplicação foi implantada em um servidor utilizando o sistema operacional Ubuntu na versão 14.04.

Para armazenamento de dados foi utilizado a distribuição *community* do MySQL na versão 5.5 e o Redis na versão 2.8.4.

Redis disponibiliza clientes para diversas linguagens de programação do mercado como ActionScript, Bash, C, C#, C++, Java, Lua, Pearl, PHP, Python, Ruby, entre outros, informações sobre mais clientes estão disponíveis no próprio site da ferramenta. Para utilização do Redis através do PHP, foi utilizado o cliente Predis [Predis 2015], que já é um cliente Redis maduro e possui algumas formas de instalação que trazem mais simplicidade nesse processo. A instalação do Predis foi realizado através do Composer [Composer 2015], um gerenciador de dependências que já se encontra na instalação padrão do *framework* Symfony [Symfony 2015].

A aplicação possui três entidades principais que se relacionam entre si de acordo com o diagrama exibido na Figura 6.

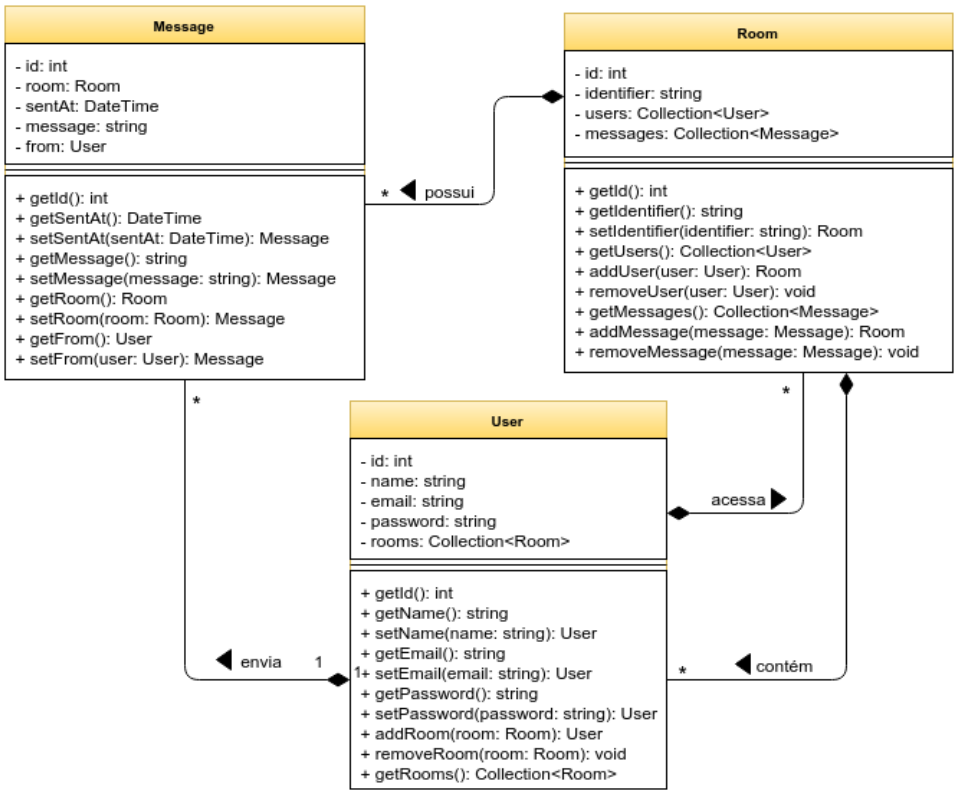

Figura 6. Diagrama de classes da aplicação utilizada no estudo de caso. Fonte: elaborado pelo autor.

A aplicação possui dois fluxos principais, o primeiro deles é a abertura da sala de conversa e pode ser observador pelo diagrama de sequência na Figura 7.

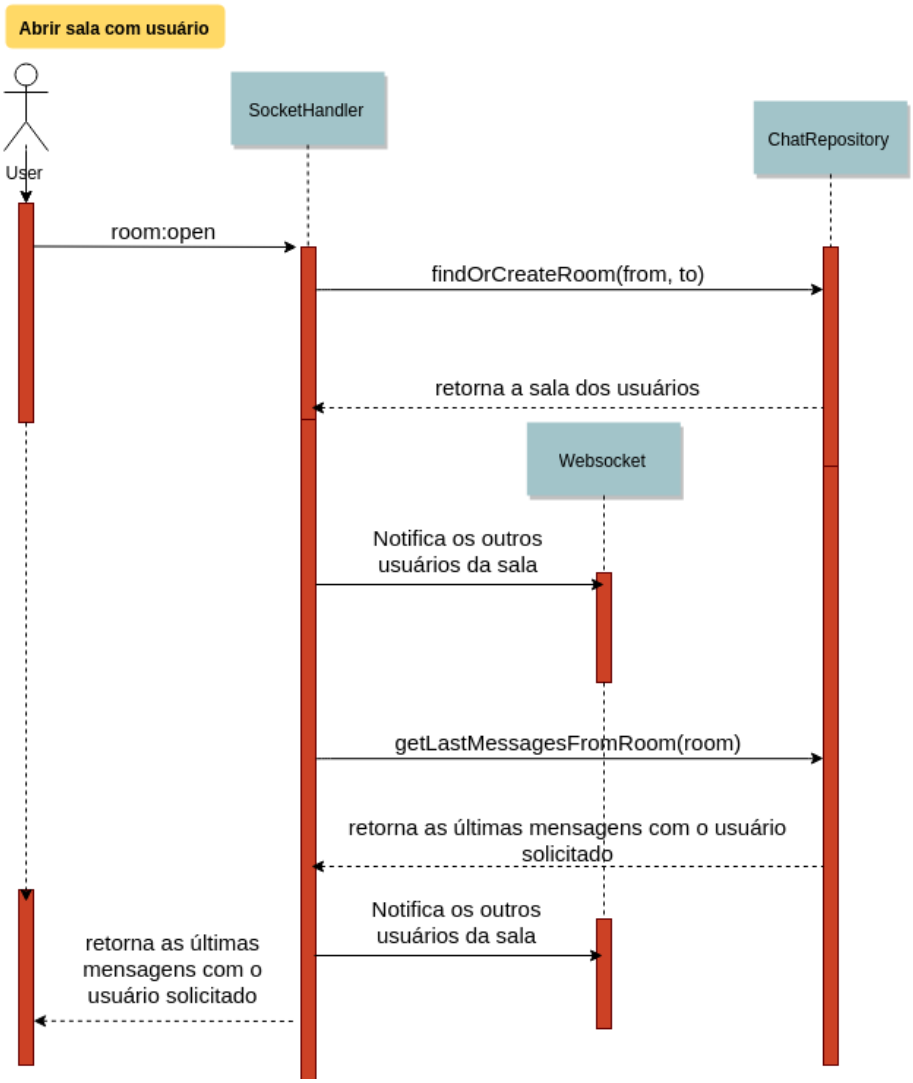

Figura 7. Diagrama de sequência de abertura de sala de conversa. Fonte: elaborado pelo autor.

Esse fluxo é iniciado pelo usuário ao selecionar um contato para a conversa. A ação de selecionar um contato notifica o servidor que um usuário está tentando criar uma sala de conversa com outro usuário. O servidor verifica se existe alguma sala previamente criada entre esses dois usuários e verifica se existe mensagens já trocadas entre os dois usuários. Depois das verificações o sistema notifica o usuário convidado com a abertura da sala, as últimas mensagens trocadas entre os usuários e notifica o usuário que enviou o convite com as últimas mensagens enviadas entre os dois.

Ao final do fluxo de abertura de sala de conversa o fluxo de envio de mensagens pode ser iniciado.

O fluxo de envio de mensagens é o segundo fluxo da aplicação e pode ser observado no diagrama de sequência da Figura 8.

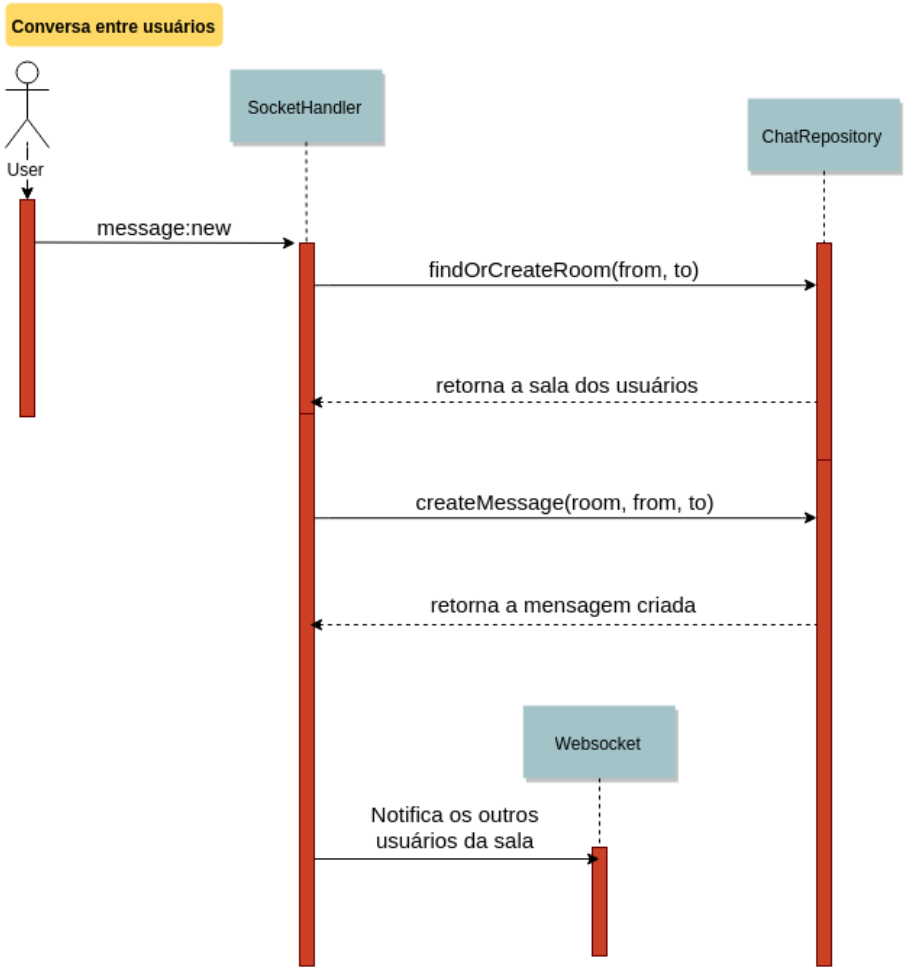

Figura 8. Diagrama de sequência de troca de mensagens entre usuários. Fonte: elaborado pelo autor.

Nesse caso, o usuário inicia o fluxo através do envio de uma mensagem à sala, que será exibida para todos os usuários.

Ao enviar a mensagem, o servidor é notificado com a mensagem e busca a sala para aquela mensagem, ao encontrar a sala, a mensagem é criada e salva utilizando o armazenamento de dados selecionado. Após a persistência, os usuários da sala são notificados da nova mensagem.

3.2 Arquitetura para o levantamento de dados

Nos dois fluxos apresentados os dados são enviados para armazenamento através da classe `ChatRepository`. Esta classe, em um primeiro momento, tem a função de armazenar os dados em banco de dados. Essa abordagem permite que todos os dados sejam acessados posteriormente, porém o custo de se fazer solicitações de leitura e gravação constantes no banco de dados é a diminuição de desempenho.

Ao atrasar no processamento de cada requisição pelo armazenamento no banco de dados, a experiência em tempo real dos usuários poderá ser comprometida ou até impossibilitada com o crescente número de mensagens e salas de conversa.

Para verificar se o armazenamento no banco de dados prejudicará o desempenho da aplicação foram criadas duas classes: `RedisChatRepository` e

`MySQLChatRepository`, ambas implementando uma interface `ChatRepository` que expõe os métodos necessários à implementação.

Os dados para verificar a diferença de desempenho entre banco de dados e cache foram adquiridos no processo de criação de mensagens apresentado na Figura 8. Através da função *microtime* do PHP é possível recuperar o tempo no padrão Unix em microssegundos, para verificar a duração da execução do armazenamento foi comparado o tempo antes da execução do armazenamento e depois da execução do armazenamento.

4. Resultados

Os dados coletados foram produzidos através de 200 blocos de requisições realizadas, cada bloco contendo 5 requisições simultâneas separadas por um intervalo de 100 milissegundos com o objetivo de simular a interação de usuários com o sistema. O tempo coletado no caso de teste leva em consideração apenas a persistência dos dados. Esse caso de teste foi realizado 10 vezes e os resultados apresentados nesse artigo são uma média dos resultados.

Tabela 1. Média do tempo de resposta das requisições em microssegundos

Quantidade de requisições	Redis (μs)		MySQL (μs)	
	Média	Total	Média	Total
1-1.000	0,00276749	2,77	0,00806953	8,07
1.001-2.000	0,00331451	3,32	0,00855559	8,56
2.001-3.000	0,00390531	3,91	0,00806647	8,07
3.001-4.000	0,00302555	3,03	0,00834436	8,35
4.001-5.000	0,00309597	3,10	0,00822646	8,23
5.001-6.000	0,00312439	3,13	0,00816170	8,17
6.001-7.000	0,00513603	5,14	0,00817872	8,19
7.001-8.000	0,00397871	3,98	0,00847566	8,48
8.001-9.000	0,00349764	3,50	0,00803599	8,04
9.001-10.000	0,00441038	4,89	0,00832083	8,33

Na Tabela 1 são exibidos os dados coletados em intervalos de 10.000 requisições, nesses dados é possível perceber que existe uma diferença de desempenho entre o armazenamento utilizando o Redis e o MySQL. Na Figura 9, são exibidos os dados das requisições de 1 até 1.000.

Figura 9. Tempo de execução em microssegundos. Fonte: elaborado pelo autor.

Na Figura 9 é possível perceber que apesar do Redis ser mais rápido na média, ele apresentou grandes picos de tempo em determinadas requisições, o que prejudica o desempenho como um todo.

O tempo total das requisições foram de cerca 36 microssegundos para Redis contra 82 microssegundos para o MySQL, esses dados demonstram que ao utilizar o armazenamento em cache e posteriormente salvar os dados no banco de dados apresentou um desempenho 2,3 vezes melhor que salvar os dados apenas no banco de dados como pode ser observado na Figura 10.

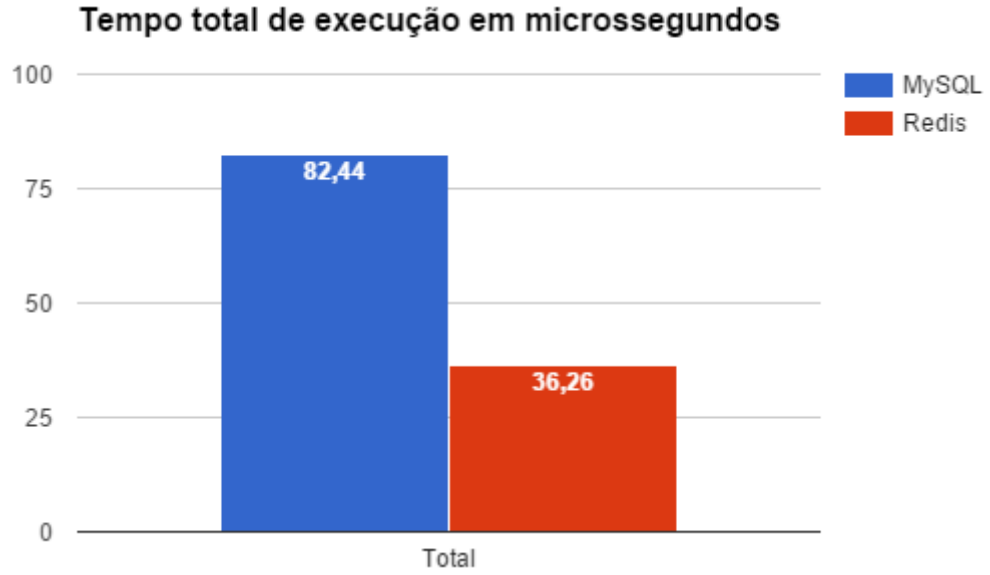

Figura 10. Tempo total de execução em microssegundos. Fonte: elaborado pelo autor

No gráfico da Figura 11, é demonstrado a porcentagem de requisições processadas em razão do tempo em microssegundos.

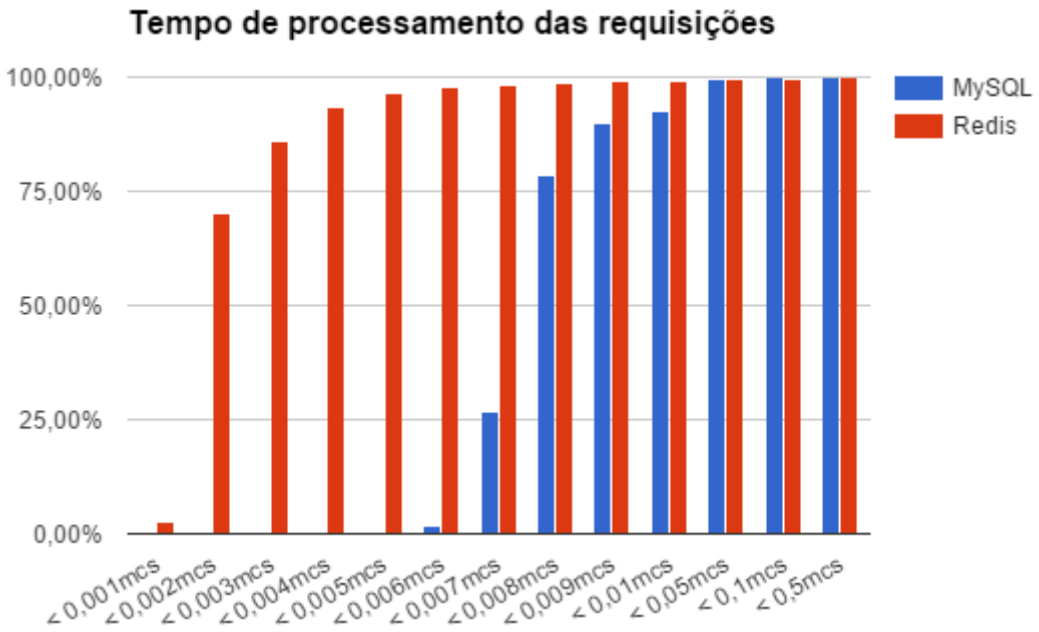

Figura 11. Requisições concluídas por tempo de processamento em microssegundos. Fonte: elaborado pelo autor.

Na Tabela 2, podem ser observados os dados da Figura 11 mais detalhados e com mais clareza.

Tabela 2. Requisições concluídas por tempo de processamento em microssegundos

Duração do processamento	Redis		MySQL	
	Qtd.	%	Qtd.	%
< 0,001 µs	283	2,83%	0	0,00%
< 0,002 µs	6.997	69,97%	0	0,00%
< 0,003 µs	8.585	85,85%	0	0,00%
< 0,004 µs	9.344	93,44%	0	0,00%
< 0,005 µs	9.650	96,50%	9	0,09%
< 0,005 µs	9.770	97,70%	158	1,58%
< 0,007 µs	9.841	98,41%	2.669	26,69%
< 0,008 µs	9.881	98,81%	7.854	78,54%
< 0,009 µs	9.889	98,89%	8.989	89,89%
< 0,01 µs	9.914	99,14%	9.259	92,59%
< 0,05 µs	9.939	99,39%	9.956	99,56%

< 0,1 μs	9.947	99,47%	9.983	99,83%
< 0,5 μs	9.999	99,99%	10.000	100%

Esses dados demonstram que a velocidade de processamento das mensagens utilizando o cache foi superior ao processamento com banco de dados, com o Redis realizando grande parte do armazenamento das mensagens em menos de 0,007 microssegundos.

Esta diferença de desempenho no MySQL ocorre devido ao armazenamento em disco rígido que é mais lento que o armazenamento em memória RAM no caso do Redis. Apesar da abordagem com o Redis ter tido um desempenho melhor, esse desempenho ainda sofreu um atraso devido ao envio dos dados ao banco de dados quando tamanho especificado para o cache foi completado.

Além do atraso com o armazenamento no banco de dados, Redis sofre atrasos também com a rede pois o protocolo de transferência de dados do Redis utiliza o protocolo TCP para o envio de mensagens [Redis] enquanto o MySQL, sendo executado em ambiente local, utiliza um *socket* Unix [MySQL], apresentando um desempenho melhor frente ao TCP/IP [Northcutt].

5. Conclusão e sugestão de estudo

Neste trabalho foi apresentado os métodos de otimização de aplicações web por meio de cache. Um estudo de caso foi desenvolvido para realizar comparações entre uma abordagem sem o uso de cache e outra utilizando o software Redis.

O estudo de caso envolveu uma aplicação de troca de mensagens entre usuários onde as mensagens deveriam ser armazenadas no banco de dados. Para garantir o desempenho do armazenamento foram realizados testes para definir o tempo gasto em cada operação em duas situações: utilização de armazenamento em cache e armazenamento diretamente no banco de dados.

Os resultados dos testes mostraram que o armazenamento diretamente no banco de dados foi mais de duas vezes mais lenta que o armazenamento utilizando cache, evidenciando a necessidade de se pensar em estratégias para melhorar o desempenho das aplicações.

Mais investigações podem ser realizadas para melhorar o desempenho em todo o ciclo de operação de uma aplicação com demanda de grande número de requisições. Como sugestão para trabalhos futuros, o uso de cache de HTTP e balanço de carga se mostram como uma sequência direta desse trabalho. Outras investigações sobre cache de imagens e alternativas para alterar os registros em cache também podem ser estudadas. O uso de cache de conteúdo dinâmico, onde a duração do mesmo deve ser associada com a natureza dos dados é também uma oportunidade de investigação.

REFERÊNCIAS

Amazon (2015) “What Is Amazon DynamoDB?”, Disponível em: <http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>. Acesso em: 01 de novembro de 2015.

"An interview with Salvatore Sanfilippo, creator of Redis, working out of Sicily", Disponível em: <http://www.eu-startups.com/2011/01/an-interview-with-salvatore-sanfilippo-creator-of-redis-working-out-of-sicily/>. Acesso em: 01 de maio de 2015.

Arpaci-Dusseau, R. e Apaci-Dusseau, A. (2015) "Operating Systems: Three Easy Peices", Arpaci-Dusseau Books, Disponível em: <http://pages.cs.wisc.edu/~remzi/OSTEP/vm-beyondphys-policy.pdf>. Acesso em: 22 de novembro de 2015.

Basho (2015) "Why Riak", Disponível em: <http://docs.basho.com/riak/latest/theory/why-riak/>. Acesso em: 01 de novembro de 2015.

Carlson, J. (2013) "Redis in Action", Manning Publications.

Composer (2015) "Documentation", Disponível em: <https://getcomposer.org/doc/>. Acesso em: 01 de novembro de 2015.

Dell (2011) "Solid State Drive vs. Hard Disk Drive Price and Performance Study", Disponível em: <http://www.dell.com/downloads/global/products/pvaul/en/ssd_vs_hdd_price_and_performance_study.pdf>. Acesso em: 22 de novembro de 2015.

Drew, M. (2008) "Measuring the Performance Effects of mod_deflate in Apache 2.2", Web Performance Inc, Disponível em: <http://www.webperformance.com/library/reports/moddeflate/>. Acesso em: 21 de novembro de 2015.

Evans, C. (2014) "Write-through, write-around, write-back: Cache explained", Computer Weekly, Disponível em: <http://www.computerweekly.com/feature/Write-through-write-around-write-back-Cache-explained>. Acesso em: 28 de setembro de 2015.

Garcia-Molina, H. e Salem, K. (1992) "Main memory database systems: An overview", IEEE Transactions on knowledge and data engineering.

Gebert, S., PRIES, R., Schlosser, D. and Heck, K. (2012) "Internet Access Traffic Measurement and Analysis", University of Wurzburg, Institute of Computer Science, Alemanha.

Github (2009) "How We Made GitHub Fast", Disponível em: <https://github.com/blog/530-how-we-made-github-fast>. Acesso em: 01 de setembro de 2015.

Heise (2010) "NoSQL im Überblick", Disponível em: <http://www.heise.de/open/artikel/NoSQL-im-Ueberblick-1012483.html>. Acesso em: 28 de agosto de 2015.

Instagram (2012) "What Powers Instagram: Hundreds of Instances, Dozens of Technologies", Disponível em: <http://instagram-engineering.tumblr.com/post/13649370142/what-powers-instagram-hundreds-of-instances>. Acesso em: 01 de setembro de 2015.

Jouppi, N. (1991) "Cache Write Policies and Performance", WRL Research Report.

Joy, P. T. e Jacob, K. P. (2012) "Cache Replacement Strategies for Mobile Data

Caching", Department of Computer Science, Cochin University of Science and Tecnology, India.

Krikorian, R. (2012) "Real-Time Delivery Architecture at Twitter", InfoQ, Disponível em: <http://www.infoq.com/presentations/Real-Time-Delivery-Twitter>. Acesso em: 01 de setembro de 2015.

Lorenzetti, P., Rizzo, L. e Vicisano, L. (2000) "Replacement policies for a proxy cache", Dipartimento di Ingegneria dell'Informazione, Università di Pisa.

Memcached (2015) "About Memcached", Memcached, Disponível em: <http://memcached.org/about>. Acesso em: 01 de novembro de 2015.

Moy, J. (1998) "OSPF Version 2", Request For Comments 2328, IETF, Disponível em: <https://tools.ietf.org/html/rfc2328>. Acesso em: 22 de novembro de 2015.

MSDN (2011) "Apresentando o Windows Server AppFabric", MSDN, Disponível em: <https://msdn.microsoft.com/pt-br/library/ee677312.aspx>. Acesso em: 01 de novembro de 2015.

MySQL. "Connecting to MySQL Server", Disponível em: <https://dev.mysql.com/doc/refman/5.5/en/connecting.html>. Acesso em: 06 de novembro de 2015.

MySQL Cache (2015) "Buffering and Caching", Disponível em: <http://dev.mysql.com/doc/refman/5.7/en/buffering-caching.html>. Acesso em: 22 de novembro de 2015.

Nanda, A. (2015) "Caching and Pooling", Disponível em: <http://www.oracle.com/technetwork/articles/sql/11g-caching-pooling-088320.html>. Acesso em: 22 de novembro de 2015.

Northcutt, B. "MySQL Connection Speed: Socket VS TCP/IP", Disponível em: <http://brandon.northcutt.net/article/MySQL+Connection+Speed%26%2358%3B+Socket+VS+TCP%26%2347%3BIP/20140425.html>. Acesso em: 06 de novembro de 2015.

Nygren, E., Sitaraman, R. e Sun, J. (2010) "The Akamai Network: A Platform for High-Performance Internet Applications", Department of Computer Science, University of Massachusetts, United States.

Pitkow, J. e Recker, M. (1994) "A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns", Georgia Institute of Technology.

Predis (2015) "Predis", Disponível em: <https://github.com/nrk/predis>. Acesso em: 01 de novembro de 2015.

Redis. Disponível em: <http://redis.io/>, Acesso em: 29 de agosto de 2015.

Redmond, E. e Wilson, J. (2012) "Seven Databases in Seven Weeks: A guide to modern databases and the NoSQL movement", Pragmatic Programmers.

Seguin, K. (2014). "The Little Redis Book", 2014.

Solid IT (2015) "DB-Engines Ranking", Disponível em: <http://db-engines.com/en/ranking>. Acesso em: 24 de outubro de 2015.

Solid IT (2015) “Method of calculating the scores of the DB-Engines Ranking”, Disponível em: <http://db-engines.com/en/ranking_definition>. Acesso em: 24 de outubro de 2015.

Symfony (2015) “Learn Symfony”, Disponível em: <http://symfony.com/doc/current/index.html>. Acesso em: 01 de novembro de 2015.