Um Estudo Experimental para Caracterização de Alocações Dinâmicas de Memória

Diego Elias D. Costa, Rivalino Matias Jr
Faculdade de Computação
Universidade Federal de Uberlândia
Uberlândia, Brasil
diegoelias@comp.ufu.br, rivalino@fc.ufu.br

***Abstract*—Dynamic memory allocation is one of the most ubiquitous operations in computer programs. In order to design effective memory allocation algorithms, it is a major requirement to understand the most frequent memory allocation patterns present in modern applications. In this paper, we present an experimental characterization study of dynamic memory allocations in seven real-world widely adopted applications. The results show several consistent allocation/deallocation patterns present in different application categories.**

Keywords—memory management; dynamic allocation; usage characterization

I. INTRODUÇÃO

Alocação dinâmica de memória é uma das operações mais onipresentes em programas de computador. Em geral, aplicações mais sofisticadas alocam e liberam, dinamicamente, porções de memória de tamanhos variados, muitas vezes, durante suas execuções. A alta taxa de ocorrência destas operações faz com que suas execuções tenham impacto significativo no desempenho de sistemas computacionais.

Em [1] é apresentado um estudo empírico comparativo de sete alocadores de memória. A comparação se deu executando uma aplicação real com cada alocador. Os resultados permitiram comparar o desempenho dos alocadores em termos de tempo de resposta, uso de memória e fragmentação de memória. Contudo, tais resultados são aplicáveis apenas para aplicações com perfis de uso de memória similares ao da aplicação usada no estudo, cujas principais características foram a predominância de alocações de tamanho menor que 64 bytes e com o maior número de alocações no início da execução. Em [2], os alocadores considerados em [1] foram novamente avaliados, contudo usando uma carga de trabalho sintética. Esta abordagem permitiu maior flexibilidade para testar os alocadores em diferentes condições experimentais, variando o tamanho das alocações, número de alocações, número de *threads* e número de processadores. Uma limitação do trabalho é que a definição dos valores destes parâmetros foi baseada, principalmente, na experiência dos autores, pois não foram encontrados trabalhos na literatura que subsidiassem esta definição por meio de estudos de caracterização do uso de memória em diferentes tipos de aplicações reais.

Objetivando suprir esta necessidade, por meio da identificação de diferentes padrões de alocação para serem usados durante a geração de cargas de trabalho sintéticas e servir de base para as pesquisas em algoritmos de alocação de memória, neste artigo é apresentado um estudo experimental de caracterização de alocações dinâmicas de memória, realizado com sete aplicações reais. As demais seções deste trabalho estão organizadas como segue. Na Seção II tem-se a descrição da metodologia, apresentando a instrumentação usada, as sete aplicações analisadas e o planejamento experimental adotado. A Seção III apresenta os resultados obtidos por meio dos experimentos de caracterização. A Seção IV apresenta as considerações finais do trabalho.

II. MATERIAIS E MÉTODOS

A. Instrumentação

Para realizar a caracterização do comportamento das aplicações em termos da alocação dinâmica de memória, se fez necessário instrumentar suas rotinas de alocação/desalocação de memória a fim de coletar dados em tempo de execução. Para tanto, optou-se por uma abordagem menos intrusiva, realizada sem modificação do código fonte da aplicação. Para isso, foi desenvolvido um *wrapper* de alocador de memória, o qual será daqui em diante chamado de *DebugMalloc*. Este intercepta e redireciona as chamadas de rotinas de alocação e desalocação de memória, realizando a coleta dos valores dos seus parâmetros e em seguida repassando-os para o alocador padrão (ver Fig. 1). No Linux, o atual alocador de memória padrão é o *ptmalloc2* [3]. Com este tipo de instrumentação foi possível coletar dados de cada rotina de alocação/desalocação, com flexibilidade e precisão, de forma menos intrusiva do que outras ferramentas, tais como *SystemTap*, *Ptrace* e *Valgrind*. Para que o *DebugMalloc* seja executado é realizada a ligação dinâmica do seu código com a aplicação, em tempo de execução, por meio da variável de ambiente LD_PRELOAD que aponta para a biblioteca compartilhada contendo o *DebugMalloc*.

Ao ser acionado, o *DebugMalloc* coleta um conjunto de dados para cada uma das operações de alocação e desalocação realizadas pelas aplicações. As Tabelas I e II apresentam os dados coletados nas operações de alocação e desalocação, respectivamente. Os dados coletados pelo *DebugMalloc* ficam armazenados na memória principal durante todo o experimento. Tal estratégia foi adotada para minimizar o efeito do acesso ao disco durante a execução das aplicações. Ao final do

experimento o *DebugMalloc* salva os dados em disco. Neste estudo, não se considerou o custo computacional da instrumentação adotada, pois o foco do trabalho foi na caracterização das alocações dinâmicas de memória das aplicações, o que independe do tempo de execução das operações.

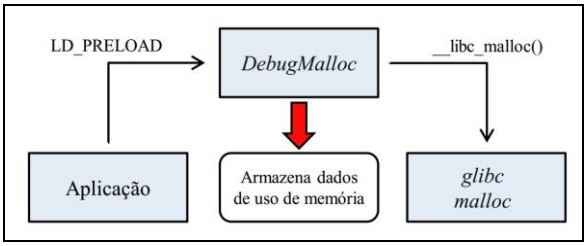

Fig. 1. Diagrama de funcionamento do *DebugMalloc*.

TABELA I. DADOS COLETADOS POR OPERAÇÃO DE ALOCAÇÃO

Dados	Descrição
Tamanho (em bytes)	Tamanho da requisição.
Tipo de operação	Rotinas de alocação requisitadas: *malloc*[a], *calloc* e *realloc*.
Tempo (em milisegundos)	Instante em que a alocação foi requisitada.
Endereço	Endereço de memória retornado pelo alocador padrão.

[a] O operador *new* chama internamente *malloc*, portanto, todo uso do *new* foi categorizado como *malloc*.

TABELA II. DADOS COLETADOS POR OPERAÇÃO DE DESALOCAÇÃO

Dados	Descrição
Tempo (em milisegundos)	Instante em que a desalocação foi requisitada.
Endereço	Endereço de memória que será liberado pela aplicação.

B. Aplicações

Neste estudo foram selecionadas sete aplicações. A escolha das aplicações seguiu os seguintes critérios:

- A aplicação deve ser amplamente usada.
- A aplicação deve ser executada no sistema operacional Linux.
- A aplicação deve ser programada nas linguagens C/C++. Tal critério se faz necessário uma vez que o *DebugMalloc* foi desenvolvido para interceptar as chamadas para o alocador padrão da *glibc* [3]; a biblioteca padrão para programas escritos em C/C++ no Linux.
- A aplicação deve utilizar o alocador padrão da *glibc*. Algumas aplicações não usam o alocador padrão e trazem o seu próprio alocador de memória. O *DebugMalloc* intercepta as chamadas para o alocador padrão.
- A aplicação deve permitir a automatização das suas principais operações. Todos os testes foram automatizados para serem replicados sem intervenção do usuário, evitando qualquer influência não controlada.

As aplicações escolhidas contemplam duas categorias de programas: *Desktop* e *Servidor*. Foram utilizadas duas aplicações da categoria *Servidor* e cinco da categoria *Desktop*:

1) MySQL: é um gerenciador de banco de dados largamente usado, com mais de 100 milhões de cópias distribuídas [4]. A versão utilizada foi a MySQL Community Server 5.6.12.

2) Cherokee: é um servidor web leve e de alto desempenho [5]. Alguns testes [6] já apontam o Cherokee como um dos melhores *web servers* em termos de desempenho, tanto para conteúdo estático quanto dinâmico. Utilizou-se neste estudo a versão estável 1.0.

3) CodeBlocks: é um ambiente integrado para desenvolvimento de software (IDE) em C, C++ e Fortran [7]. Desenvolvido desde 2005, a versão utilizada neste estudo foi a 13.2.

4) VLCPlayer: é um reprodutor multimídia multi-plataforma [8]. Com mais de 17 anos de desenvolvimento, a versão utilizada neste estudo foi a 2.1.4.

5) Octave: é uma linguagem interpretada de alto-nível para realizar computações numéricas [9]. Este provê desde soluções numéricas até funcionalidades de manipulação de gráficos, semelhante ao Matlab. Vem sendo desenvolvido desde 1988 e a versão utilizada foi a 3.8.1.

6) Inkscape: é um editor multiplaraforma de gráficos vetoriais [10]. Com mais de 10 anos de desenvolvimento, a versão utilizada neste estudo foi a 0.48.4.

7) Lynx: é um navegador web baseado em interface textual [11]. Vem sendo desenvolvido desde 1992 e a versão utilizada neste estudo foi a 2.8.7rel.2.

C. Planejamento Experimental

A caracterização das alocações dinâmicas foi feita utilizando um cenário típico de uso, específico para cada aplicação. Cada teste de caracterização foi replicado trinta vezes com o propósito de minimizar a influência de erros experimentais. Os dados analisados foram obtidos através da média das replicações. A seguir uma descrição dos cenários de teste de cada aplicação.

1) MySQL: Para caracterizar o uso da memória do MySQL foi usado o banco de dados de teste *Sakila-Database* [12]. Este é um banco de dados funcional que simula uma locadora de filmes de grande porte, possui 22 tabelas e utiliza todas as principais estruturas de dados e buscas do MySQL, tais como *views, stored procedures e triggers*. Para realizar as consultas foi usado o MySQLSlap [13], uma aplicação de teste que

realiza um conjunto pré-determinado de operações no banco de dados de forma automatizada. O cenário de teste consiste na criação do banco de dados *Sakila* e então é realizado um conjunto de consultas e alterações no banco de dados por 50 clientes simultâneos. Cada cliente realiza 36 operações entre consultas e alterações. Ao final do experimento, o banco de dados de teste é apagado do disco.

2) Cherokee: Para caracterizar o uso de memória do Cherokee foi utilizado o aplicativo Apache-Bench [14], o qual permite automatizar o acesso a páginas web. Foi criado um cenário de teste com vinte clientes realizando 50 acessos simultâneos à página administrativa do Cherokee.

3) CodeBlocks: O caso de teste escolhido para o CodeBlocks foi a inicialização do programa e o carregamento de uma área de trabalho contendo o código do projeto do próprio CodeBlocks. Foi usado o código da versão 13.12, sendo que o código e artefatos ocupam aproximadamente 20MB de espaço em disco; uma aplicação de grande porte.

4) VLCPlayer: Para caracterizar o uso de memória do VLCPlayer foram criados dois casos de teste. No primeiro foi realizada a execução ininterrupta de uma faixa de áudio (5:34 minutos). No segundo foi feita a execução de um curta-metragem em vídeo (9:56 minutos).

5) Octave: O caso de teste escolhido foi a execução do método de eliminação de Gauss para escalonar uma matriz tridiagonal de ordem 50. Tal algoritmo é comumente utilizado para resolução de problemas lineares [15], uma das principais funcionalidades do Octave.

6) Inkscape: O caso de teste escolhido foi a inicialização completa da aplicação seguida do carregamento de uma imagem com a resolução 1920x1080 e com tamanho de 280KB para edição.

7) Lynx: Neste caso foi realizado o acesso a um conjunto de cinco páginas. Para escolher o conjunto de páginas usadas, foram selecionadas as cinco páginas mais acessadas da Internet segundo o *Alexa Rank* [16].

Todos os experimentos foram realizados em uma bancada de teste composta de um computador multicore (Intel Core i5 2410M), com 6GB de memória RAM, com Linux (Kernel 3.11.6-4-desktop) e distribuição OpenSuse 13.1. A Fig. 2 apresenta a topologia de processador e memória *cache* do computador utilizado.

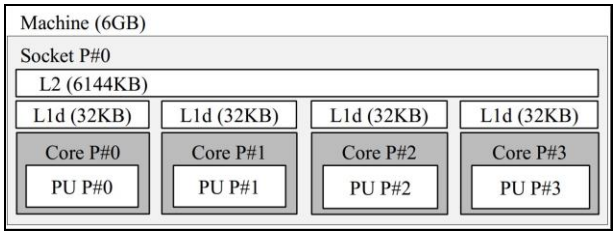

Fig. 2. Topologia do processador usado nos experimentos.

III. ANÁLISE DOS RESULTADOS

A. Total de Memória Alocada

Observou-se uma considerável diversidade na quantidade de memória alocada e no número de alocações feitas pelas aplicações (ver Tabela III). A memória total alocada por cada aplicação variou entre 9MB com 30.000 alocações (Lynx) e 2,4GB (CodeBlocks) alocados em mais de 12,4 milhões de alocações. Neste estudo, a aplicação que mais alocou memória foi também a que mais vezes requisitou o alocador. No entanto, o caso de teste com o Inkscape mostrou também que embora este tenha requisitado 12,1 milhões de alocação, a sua memória total utilizada foi apenas de 564 MB, devido ao seu baixo tamanho médio de alocação.

B. Tamanhos Alocados

Neste estudo foi analisado o tamanho das alocações de cada aplicação, e os resultados obtidos na distribuição dos tamanhos podem ser observados nas Fig. 3-10. Note que a maior parte das alocações está concentrada em uma faixa bem definida de tamanhos. Todas as aplicações *Desktop* testadas obtiveram a maioria das alocações com tamanhos de até 100 bytes, enquanto as aplicações MySQL e Cherokee, ambas do tipo *Servidor*, apresentaram a maior parte das alocações com tamanhos 1144 e 264 bytes, respectivamente. Vale ressaltar que o teste feito com o VLCPlayer (áudio) também obteve uma considerável parcela das alocações com tamanhos de 1000 até 1230 bytes.

As aplicações alocaram em média 2336 tamanhos distintos em seu tempo de execução, no entanto, a maior parte das alocações está concentrada em um grupo de dez tamanhos. Em média, os dez tamanhos mais alocados representam 75,2% do total das alocações, variando de 50% com o Lynx até 92% com o Cherokee (ver Fig. 11).

Geradores de carga sintética, para testes de alocadores, tendem a tratar o tamanho da alocação de memória de forma aleatória, alocando tamanhos randômicos para cada requisição [2] ou fixando um tamanho único para as alocações [17][18]. O comportamento observado neste estudo sugere que uma junção destas duas práticas oferece uma carga de trabalho mais próxima de uma aplicação real, ao se fixar um pequeno conjunto de tamanhos como a maior parte das alocações e aleatorizar, de forma homogênea, as demais alocações de memória.

TABELA III. ALOCAÇÕES POR APLICAÇÃO

Aplicação	Número de Alocações	Memória Alocada (MB)	Tamanho Médio Alocado (Bytes)
MySQL	467.348	686	1.467,40
Cherokee	31.727	14	420,13
CodeBlocks	12.412.302	2.493	200,92
VLCPlayer (áudio)	138.747	335	2.410,82
VLCPlayer (vídeo)	1.513.223	1.885	1.245,42
Octave	579.606	100	172,07
Inkscape	12.172.463	564	46,31
Lynx	31.303	9	268,51

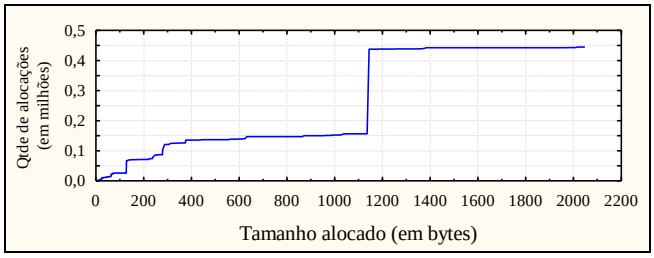

Fig. 3. Distribuição dos tamanhos alocados para o MySQL.

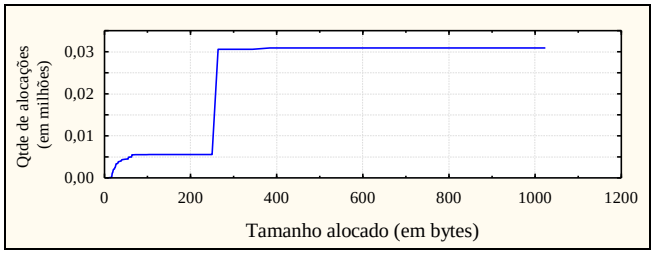

Fig. 4. Distribuição dos tamanhos alocados para o Cherokee.

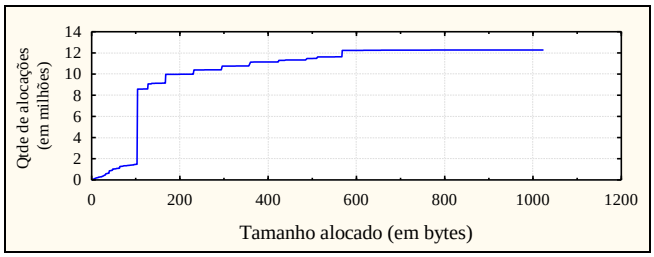

Fig. 5. Distribuição dos tamanhos alocados para o CodeBlocks.

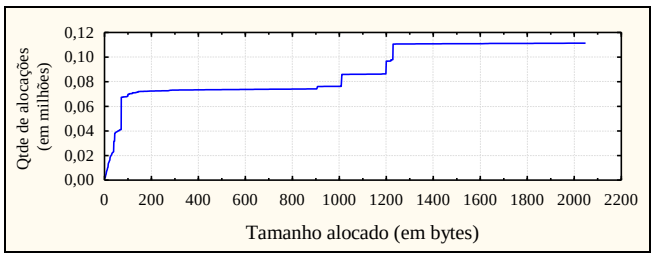

Fig. 6. Distribuição dos tamanhos alocados para o VLCPlayer (áudio).

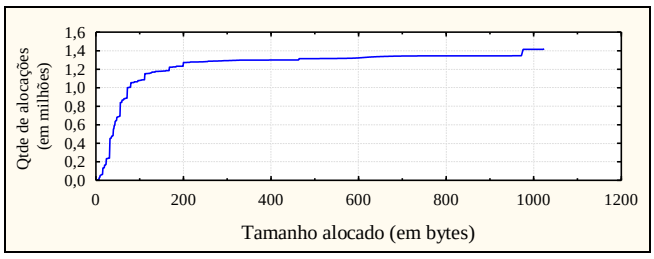

Fig. 7. Distribuição dos tamanhos alocados para o VLCPlayer (vídeo).

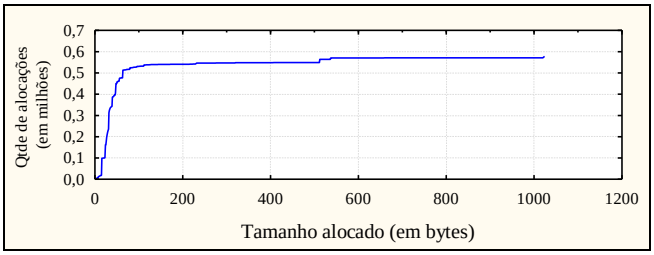

Fig. 8. Distribuição dos tamanhos alocados para o Octave.

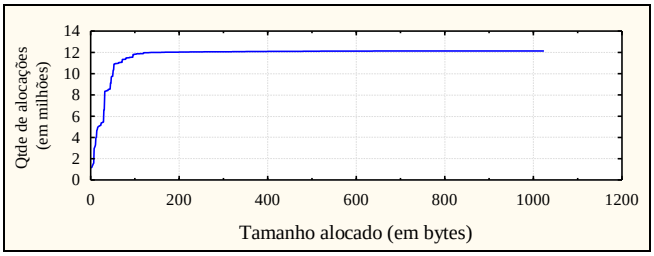

Fig. 9. Distribuição dos tamanhos alocados para o Inkscape.

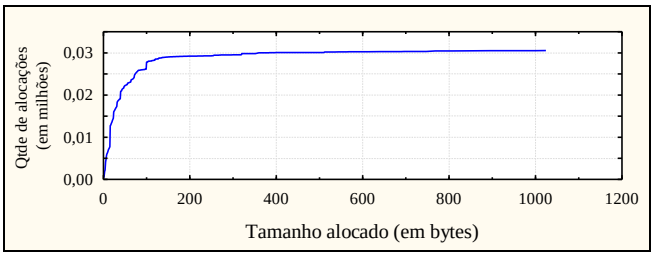

Fig. 10. Distribuição dos tamanhos alocados para o Lynx.

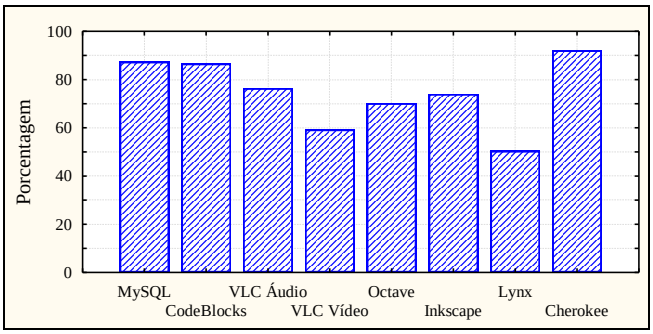

Fig. 11. Porcentagem dos dez tamanhos mais alocados por aplicação.

C. *Liberação de Memória*

A liberação ou desalocação de memória é um fator muito importante no comportamento de uma aplicação, do ponto de vista da alocação dinâmica de memória. A Tabela IV apresenta o número de desalocações realizadas e o percentual das desalocações com ponteiros nulos. A desalocação de ponteiros nulos é um defeito (*bug*) de software e ocorre quando a aplicação tenta desalocar uma porção de memória que foi previamente liberada ou que não havia sido alocada. Não é incomum isso provocar uma falha na aplicação. Alocações não desalocadas não foram consideradas neste estudo, pois alguns casos de teste foram interrompidos antes de sua finalização.

Os resultados mostraram que o banco de dados MySQL foi a aplicação que mais realizou desalocações com ponteiros nulos, representando quase 12% de suas desalocações, seguido pelo VLCPlayer (áudio) com 2,37%, CodeBlocks com 2,27% e o VLCPlayer (vídeo) com 2,05%. As demais aplicações obtiveram percentuais de desalocações nulas abaixo de 1%.

TABELA IV. DESALOCAÇÕES POR APLICAÇÃO

Aplicação	Número de Desalocações	% de Desalocações Nulas
MySQL	493.965	11,80
Cherokee	29.017	0,02
CodeBlocks	11.001.975	2,27
VLCPlayer (áudio)	120.051	2,37
VLCPlayer (vídeo)	1.397.594	2,05
Octave	464.206	0,96
Inkscape	11.430.033	0,11
Lynx	14.234	0,84

D. *Rotinas de Alocação*

Foram analisadas as três principais rotinas usadas para alocação dinâmica de memória: *malloc*, *calloc* e *realloc*. Cada rotina realiza a alocação da memória de uma forma específica e seu uso depende da forma como a memória será utilizada, variando de acordo com a necessidade de cada aplicação. Lembrando que o operador *new* chama a *malloc*.

Foi possível notar que em todas as aplicações a rotina mais usada foi a *malloc*, com média de 87,9% do total das alocações, seguida pela *realloc* com 6,07% e *calloc* com 6,03%. De fato, apenas as aplicações VLCPlayer e Lynx utilizaram as outras funções com maior frequência (ver Tabela V).

TABELA V. PORCENTAGEM DO USO DAS ROTINAS DE ALOCAÇÃO

Aplicação	*malloc*	*realloc*	*calloc*
MySQL	99,91	0,06	0,03
Cherokee	87,49	12,38	0,14
Code Blocks	92,51	7,18	0,31
VLCPlayer (áudio)	75,40	4,71	19,89
VLCPlayer (vídeo)	79,72	5,49	14,79
GNU Octave	99,71	0,08	0,21
Inkscape	95,61	2,93	1,46
Lynx	72,84	15,72	11,44

E. *Tempo de Retenção*

O tempo de retenção de um bloco de memória alocado é o tempo entre a sua alocação e desalocação. Este tempo é um fator muito importante para a análise e projeto de alocadores de memória, uma vez que ele tem influência na fragmentação da memória, um dos principais problemas no uso da alocação dinâmica [19]. Portanto, caracterizar esse tempo é útil no estudo dos padrões de alocação de memória. Para analisar o tempo de retenção dos blocos de memória alocados pelas aplicações investigadas, os tempos de retenção de cada bloco foram classificados em três categorias:

1) Alocações de curta duração: Alocações cujo tempo de retenção é de até 100 milissegundos.

2) Alocações de média duração: Alocações cujo tempo de retenção está entre 100 milissegundos e 1 segundo.

3) Alocações de longa duração: Alocações cujo tempo de retenção é maior do que 1 segundo. Estas alocações representam, na sua maioria, estruturas de dados que permanecem alocadas durante toda a execução da aplicação.

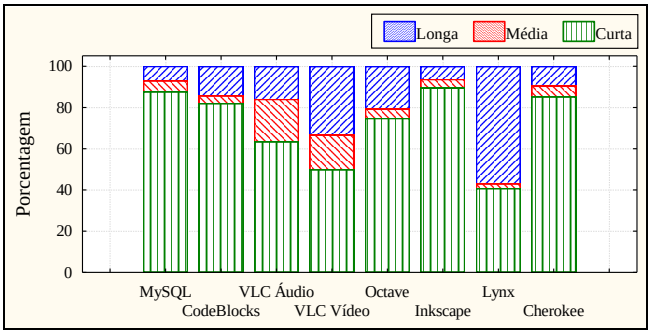

Fig. 12. Proporção das alocações de curta, média e longa durações.

Os resultados mostraram que, em média, 71,6% foram *alocações de curta duração*, enquanto 7,85% foram classificadas como *média duração*, ou seja, quase 80% das alocações são desalocadas em até 1 segundo (ver Fig. 12). No caso das aplicações *Servidor*, essa média aumenta para 86%. Esse resultado mostra que a maioria das alocações é para uso temporário, ou seja, tempo de retenção curto.

As *alocações de longa duração* representam em média 20,51% e constituem em sua maioria de estruturas de dados utilizadas durante a maior parte do tempo de execução das aplicações. Ainda de acordo com o comportamento das *alocações de longa duração* durante a execução, as aplicações foram analisadas e categorizadas em dois padrões distintos de comportamento:

1) Pico: A aplicação realiza a maior parte das *alocações de longa duração* em um conjunto pequeno e determinado de operações, evidenciado por um pico nas alocações não desalocadas. Fora da área do pico, as alocações não desalocadas se mantém em um nível aproximadamente constante, mostrando que a maior parte das alocações após o pico são *alocações de curta ou média duração.*

2) Uso crescente: A aplicação incrementa as *alocações de longa duração* gradativamente, evidenciado pela curva crescente de alocações não desalocadas. Neste caso, parte das alocações realizadas ao longo do experimento não é desalocada imediatamente, aumentando o uso de memória durante o seu período de execução.

Pela grande quantidade de operações, as análises foram realizadas quantificando as alocações não desalocadas agrupadas a cada 1000 operações de alocação/desalocação. De todas as aplicações analisadas, quatro se enquadraram no padrão *Uso crescente*, a saber: CodeBlocks (Fig. 13), Lynx (Fig. 14), Octave e VLCPlayer (vídeo). As outras quatro se enquadraram no padrão *Pico*: o VLCPlayer (áudio) (Fig. 15), MySQL (Fig. 16), Cherokee e Inkscape.

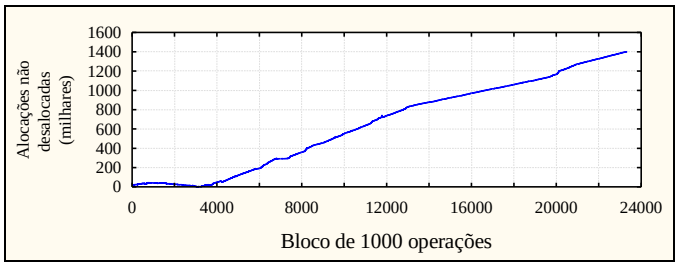

Fig. 13. Alocações não desalocadas agrupadas por blocos de 1000 operações (alocações/desalocações) da aplicação CodeBlocks.

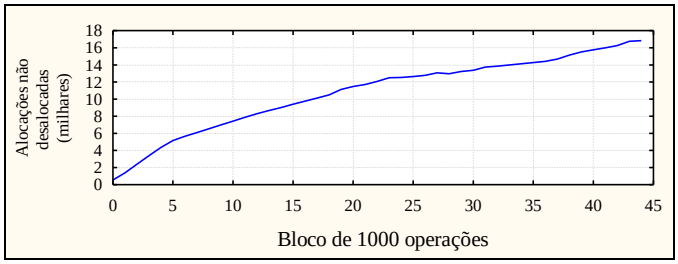

Fig. 14. Alocações não desalocadas agrupadas por blocos de 1000 operações (alocações/desalocações) da aplicação Lynx.

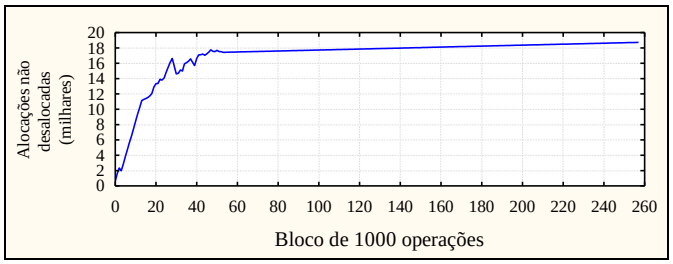

Fig. 15. Alocações não desalocadas agrupadas por blocos de 1000 operações (alocações/desalocações) da aplicação VLCPlayer (áudio).

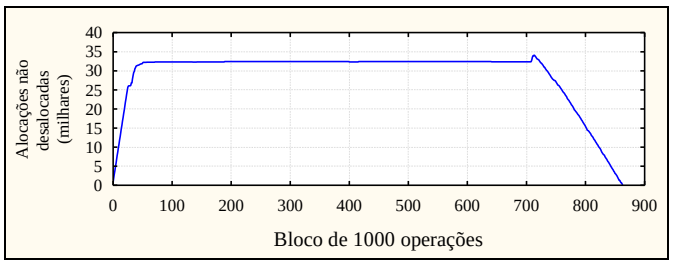

Fig. 16. Alocações não desalocadas agrupadas por blocos de 1000 operações (alocações/desalocações) da aplicação MySQL.

IV. Conclusões

Este estudo apresentou uma caracterização experimental do uso da alocação dinâmica de memória em sete aplicações reais. Observa-se que diferentes aplicações apresentaram padrões similares de alocação de memória: mais de 75% das alocações concentraram-se em um grupo de dez tamanhos diferentes. Além disso, todas as aplicações *Desktop* avaliadas realizaram, predominantemente, alocações menores do que 100 bytes, enquanto as duas aplicações *Servidor*, MySQL e Cherokee, apresentaram a maioria das alocações de 1144 e 264 bytes, respectivamente. A rotina de alocação mais acionada pelas aplicações foi a *malloc*, em 87,9% dos casos, seguida da *realloc* (6,07%) e *calloc* (6,03%). Com relação ao tempo de retenção dos blocos alocados, 71,6% dos blocos foram desalocados em até cem milissegundos e 20,5% levaram mais de um segundo para serem liberados (*alocações de longo prazo*). Dois padrões distintos de uso de memória foram observados através das alocações não desalocadas: o padrão *Uso Crescente*, observado nas aplicações CodeBlocks, Lynx, Octave e VLCPlayer (vídeo), e o padrão *Pico*, observado nas aplicações VLCPlayer (áudio), MySQL, Cherokee e Inkscape.

Os resultados deste trabalho têm sido utilizados para configurar diferentes padrões de alocação de memória durante a geração de cargas de trabalho sintéticas, as quais estão sendo usadas para investigar o comportamento de diferentes algoritmos de alocação de memória. Este estudo faz parte dos trabalhos de desenvolvimento de um novo alocador de memória sendo atualmente realizado pelos autores.

Referências

[1] T. B. Ferreira, R. Matias Jr., A. Macedo, e L. B. Araujo. “An experimental study on memory allocators in multicore and multithreaded applications,” In Proceedings of International Conference on Parallel and Distributed Computing, Applications and Technologies, Gwangju, pp. 92 - 98, Oct 2011.

[2] D. E. Costa, M. Fernandes, R. Matias Jr., e L. B. Araujo. “Análise teórico-experimental de algoritmos de alocação de memória,” In Proceedings of 3th Brazilian Symposium on Computing Systems Engineering, Niterói, Nov 2013.

[3] W. Gloger, “Ptmalloc”, http://www.malloc.de/en/

[4] MySQL, “MySQL”, http://www.mysql.com/

[5] A. L. Ortega, “Cherokee”, http://cherokee-project.com/

[6] A. H. Barea, “Analysis and evaluation of high performance web servers”, M.S. thesis, EETAC, UPC, Barcelona, 2011.

[7] N. Elah, “Code::Blocks”, http://www.codeblocks.org/

[8] VideoLAN, “VLC Media Player”, http://www.videolan.org/vlc/

[9] GNU, “GNU Octave”, http://www.gnu.org/software/octave/

[10] Inkscape, “Inkscape”, http://www.inkscape.org/pt/

[11] Lynx, “Lynx Web Browser”, http://lynx.isc.org/

[12] MySQL, “Sakila Sample Database”, http://dev.mysql.com/doc/sakila/en/

[13] MySQL, “MySQLSlap: Emulation Client”, http://dev.mysql.com/doc/refman/5.1/en/mysqlslap.html

[14] Apache Software Foundation, “Apache HTTP server benchmarking tool” , http://httpd.apache.org/docs/2.2/programs/ab.html

[15] G. D. Smith, “Solving Linear Problems: Exact Methods,” in Numerical Solution of Partial Differential Equations, 3rd ed. Oxford: OUP, 1986, ch. 4, sec. 4.3, pp. 119–122

[16] Alexa, “The top 500 sites on the web”, http://www.alexa.com/topsites

[17] E.D. Berger, K.S. McKinley, R.D. Blumofe, e P.R.Wilson, "Hoard: a scalable memory allocator for multithreaded applications", In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, v.28:5, p.117-128, 2000

[18] M. M. Michael, “Scalable Lock-Free Dynamic Memory Allocation”, Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation, p. 35-46, New York, 2004

[19] P. R. Wilson, M. S. Johnstone, M. Neely, e D. Boles, “Dynamic Storage Allocation: A Survey and Critical Review”, In Proceedings of the International Workshop on Memory Management, p.1-116, London, 1995.