Caracterização do perfil de consumo de recursos de programas binários utilizando a técnica DAMICORE

Alternative Title: Resource consumption profile characterization of binary executables programs using DAMICORE technique

Renê de Souza Pinto
Instituto de Ciências Matemáticas e de Computação
Av. Trabalhador São Carlense, 400, Caixa Postal 668
São Carlos, SP
rene@icmc.usp.br

Alexandre C. B. Delbem
Instituto de Ciências Matemáticas e de Computação
Av. Trabalhador São Carlense, 400, Caixa Postal 668
São Carlos, SP
acbd@icmc.usp.br

Francisco José Monaco
Instituto de Ciências Matemáticas e de Computação
Av. Trabalhador São Carlense, 400, Caixa Postal 668
São Carlos, SP
monaco@icmc.usp.br

RESUMO

O perfil de consumo de recursos de um software é uma informação importante que pode ser utilizada para diversos fins: em escalonadores, balanceadores de carga, sistemas de Qualidade de Serviço (QoS), entre outros. Técnicas convencionais para avaliar o perfil de carga de softwares são baseadas em análise estática do código fonte ou através de aferições obtidas durante a execução dos mesmos. Por um lado, muitas vezes não há disponibilidade do código fonte da aplicação em análise, por outro, promover aferições da execução de um processo requer cuidados especiais para minimizar o impacto da aferição no sistema. Este artigo apresenta o uso da técnica de mineração de dados DAMICORE para promover a caracterização do perfil de consumo de programas binários. A técnica foi aplicada com sucesso em um conjunto composto por 80 programas binários sistematicamente selecionados.

Palavras-Chave

DAMICORE, Clusterização, Executáveis, Caracterização

ABSTRACT

Resource consumption profile of a computer program is a relevant information with a wide scope of application such as schedulers, load balancers, Quality of Service (QoS) systems, among others. Conventional techniques available for this purpose include static source code analysis and profile matching based on runtime execution measurements. On the one hand, source code can't often be available for analysis, but on the other hand, make measurements on an execution system requires special precautions to minimize the overhead. This paper presents the use of DAMICORE data mining technique to promote the characterization of binary computer programs by their resource consumption profile. The technique was successfully applied over a dataset composed by 80 binary programs carefully selected.

SBSI 2017 June 5^{th} – 8^{th}, 2017, Lavras, Minas Gerais, Brazil

CCS Concepts

•**Information systems** → *Clustering;*

Keywords

DAMICORE, Clustering, Executables, Characterization

1. INTRODUÇÃO

O consumo de recursos do sistema por um software durante sua execução é uma informação importante que pode ser utilizada para diversos fins, tais como algoritmos de escalonamento, balanceamento de carga, planejamento de recursos, análise e eficiência energética[19] e mecanismos de alocação elástica e dinâmica com Qualidade de Serviço (QoS)[7]. No contexto de sistemas computacionais, um processo em execução demanda recursos da plataforma em que é executado tais como consumo de CPU, Entrada/Saída (E/S), Memória e quaisquer outros recursos providos pela mesma. Mesmo que o comportamento de um programa seja ditado pela sua entrada, o conjunto de instruções que o compõe podem caracterizar seu perfil de execução, permitindo por exemplo determinar se é uma aplicação que executa tarefas em disco (E/S), muitos ou poucos acessos à memória, operações de ponto flutuante ou instruções que promovam o uso intensivo de CPU. Métodos predominantes para caracterização de perfil de aplicações são inseridos basicamente em duas categorias: identificação ou inferência. A primeira corresponde as técnicas baseadas em caixa-preta que promovem a análise de dados da aplicação obtidos através de aferição, ou seja, monitorando-se a execução do aplicativo e avaliando diretamente os parâmetros de interesse no sistema. Entretanto, fazer a aferição requer adequada instrumentação do sistema para reduzir o impacto sobre o mesmo. Já as técnicas baseadas em inferência utilizam abordagens alternativas à aferição, sendo uma das mais utilizadas a análise estática

do código fonte da aplicação. Entretanto, além da disponibilidade do código fonte ser necessária, códigos grandes e complexos tornam a análise difícil, especialmente para avaliar todas as possibilidades de fluxo dos dados durante a execução do programa. Mesmo com a possibilidade da aplicação em códigos binários[11], por exemplo em *Bytecode* Java, a análise estática é em sua maioria voltada para códigos fonte, dependendo da linguagem de programação adotada, além de não considerarem otimizações promovidas pelo compilador e presentes somente no código binário final.

1.1 Objetivos

Este trabalho tem como objetivo apresentar a utilização da técnica de mineração de dados DAMICORE como uma abordagem alternativa para caracterização e análise do perfil de programas executáveis utilizando somente seus respectivos códigos binário. Esta abordagem difere-se de trabalhos existentes nos principais aspectos: somente o código binário é necessário para a análise; o DAMICORE não necessita de um conhecimento semântico dos dados a serem analisados, portanto não é necessário um conhecimento a priori das aplicações para determinar correlações entre código e desempenho; o resultado gerado é uma filogenia que descreve relações hierárquicas de similaridade entre os programas avaliados.

2. TRABALHOS RELACIONADOS

No contexto de consumo de recursos, diversos trabalhos estudam diferentes abordagens, desde métricas dinamicamente identificadas para otimização adaptativa[1, 2, 3], até técnicas que identificam cargas de trabalho através de simulação[9, 23, 22]

Balasubramonian et al.[1, 2] explora mecanismos para reconfigurar dinamicamente uma micro-arquitetura de acordo com as necessidade da aplicação. A identificação de mudanças no comportamento do programa em tempo de execução é feita com base nas instruções e acesso à memória. Dhodapkar e Smith[3] estendem o algoritmo utilizado em [1] para compor um conjunto de "assinaturas" de instruções de aplicações criando um mecanismo baseado em tabela, permitindo escolher a melhor configuração a ser utilizada no hardware reconfigurável a partir da caracterização da aplicação com base na comparação de assinaturas de execuções anteriores.

A técnica de mineração de dados DAMICORE vem sendo utilizada com sucesso em diversas áreas. FERREIRA et. al.[5] utilizam a técnica para diagnóstico da doença Greening no Citrus. Já Moro et. al.[16] utilizam a ténica para mineração de dados em ambientes virtuais de Ensino/Aprendizagem. Outros trabalhos abordam a técnica em sistemas reconfiguráveis (FPGA) e compiladores[17, 24, 14, 13].

3. DAMICORE

DAMICORE (DAta MIning of Code REpositories)[21] é um arcabouço teórico composto pela combinação de várias técnicas provindas de diversos campos, tais como Teoria da Informação, Bioinformática e Redes Complexas. Esta técnica pode ser aplicada a quaisquer conjuntos de dados, sem a necessidade do conhecimento da semântica dos mesmos, sendo capaz de encontrar relações hierárquicas entre os dados analisados. Dado um repositório de objetos de dados a serem analisados, o DAMICORE é aplicado através de três passos: o primeiro passo consiste na construção de uma matriz de distâncias que compara par à par cada um dos objetos do repositório levando em consideração uma métrica de similaridade; o segundo passo consiste na conversão da matriz de distâncias em uma rede que conecta os objetos de acordo com suas respectivas similaridades; o ultimo passo consiste na aplicação de um algoritmo de detecção de comunidades para encontrar grupos de elementos semelhantes na árvore gerada no passo anterior. Os três algoritmos utilizados originalmente nestes processos são: *Normalized Compression Distance* (NCD)[10] para a construção da matriz de distâncias; Neighbor Joining (NJ)[4] para produzir uma árvore filogenética a partir da matriz de distâncias; e o método Fast Newman (FN)[18] para detecção de comunidades na árvore gerada.

A NCD é uma aproximação computável da complexidade de Kolmogorov, sendo a distância entre dois elementos é definida pela equação 1.

$$NCD_z(a,b) = \frac{C_z(ab) - min\{C_z(a), C_z(b)\}}{max\{C_z(a), C_z(b)\}} \quad (1)$$

Os termos a e b denominam os dois elementos a serem comparados, sendo ab a concatenação de ambos. $C_z(x)$ representa o tamanho da versão compactada do elemento x obtido através de um certo algoritmo de compressão Z. Considerando um compressor ideal e dois arquivos idênticos, $C_z(ab) = C_z(a) = C_z(b)$, logo, $NCD_z(a,b) = 0$, enquanto para dois arquivos totalmente diferentes, $C_z(ab) = C_z(a) + C_z(b)$ e logo $NCD_z(a,b) = 1$. Dado que a compressão dos dados é realizada em nível de cada byte, o método DAMICORE torna-se indiferente quanto à semântica dos dados, podendo ser aplicado em qualquer tipo de repositórios: textos, imagens, áudio, códigos fonte, programas binários, pois todos são representados como sequências de bytes. Assim, o cálculo da NCD é o passo que garante essa independência semântica.

Uma vez obtida a matriz de distâncias, a aplicação do algoritmo NJ produz uma árvore filogenética dos objetos do repositório permitindo a visualização de relações antes não evidenciadas na matriz de distâncias. O método é completado através da aplicação do algoritmo FN para detecção de agrupamentos (comunidades) dos elementos semelhantes na árvore, destacando as relações mais fortes entre os objetos do repositório.

4. EXPERIMENTOS

Dentre diversas possibilidades de recursos de sistema a serem analisados, este trabalho foca-se em dois notadamente relevantes: CPU e E/S (operações de Entrada/Saída), ambos são frequentemente considerados como gargalos para diversas aplicações em cenários reais[8, 20, 12]. Entretanto, esta abordagem pode ser estendida para outros recursos de sistema, tais como memória, energia, rede, etc. Para execução dos experimentos foi composto um conjunto de 80 programas obtidos de diferentes fontes, sendo 40 programas caracterizados como *CPU-Intensive*, pois fazem predominantemente uso de CPU, e 40 programas caracterizados como *IO-Intensive*, pois fazem predominantemente operações de E/S. Todos os programas foram obtidos com seus respectivos códigos fonte (Linguagem C) e compilados com o mesmo compilador e opções (*flags*) de otimização. Para assegurar que o perfil de cada programa selecionado pertencia realmente a sua respectiva classificação, uma extensa análise

dinâmica foi executada através de *benchmarks* e aferições. Posteriormente, o DAMICORE foi aplicado sobre todo o repositório composto pelos programas binários e sobre amostras obtidas de forma aleatória a partir do repositório original. Para avaliar o impacto de diferentes compiladores e *flags* de compilação na aplicação do método, uma análise estatística também foi executada com repositórios compostos pelos programas binários compilados com os compiladores *gcc*, *icc* e *llvm*, em cada uma das principais *flags* de otimização: *-O1*, *-O2*, *-O3* e *-Os*.

4.1 Seleção de programas

Para compor o repositório de programas binários foram executadas pesquisas em bases de dados científicas, trabalhos acadêmicos e outras fontes para encontrar programas que implementassem algoritmos cujo perfil se enquadrasse em uma classificação *CPU-Intensive* (algoritmos que majoritariamente executam somente operações e consumo de CPU) ou *IO-Intensive* (algoritmos que majoritariamente executam operações de Entrada/Saída).

Os programas selecionados foram obtidos de trabalhos acadêmicos, utilitários de software presentes em sistemas operacionais de produção (tais como Linux), ferramentas específicas de *benchmark*, além de alguns códigos que foram implementados para exibirem determinado comportamento específico.

Para o grupo de programas *CPU-Intensive*, 13 algoritmos clássicos foram selecionados de [15], incluindo série de Fibonacci, *Bubble sort* para ordenação de vetores, FDCT e Sobel para processamento de imagens (filtros), ADPCM para processamento digital de sinais, e outros algoritmos de largo processamento vetorial, juntamente com *benchmarks* com operações de multiplicação de 32 bits e processamento de imagens desenvolvidos pela Texas Instruments. Também compõem o grupo uma versão variante do clássico *benchmark* de CPU *Wheatstone*, geradores de fractais, um gerador de números primos, um algoritmo de processamento de árvores rubro negras, algoritmos de compressão[6], algoritmos de stress de CPU[25] e alguns programas implementados pelo autores, sendo cinco algoritmos de ordenação e quatro algoritmos de métodos numéricos (Bisseção, Gauss, Simpson e Método dos Mínimos Quadrados).

O grupo de programas *IO-Intensive* é composto por 6 *benchmarks* de disco providos pelo projeto Linux Kernel Performance[1], 4 testadores de leitura e escrita em disco, *benchmarks* e estressadores de disco Spew e IOthrash, além de utilitários distribuídos com sistemas operacionais derivados do Unix (como o Linux) e relacionados com operações em arquivos e sistemas de arquivos (e2fsprogs, cpio, duff, dupedit, findutils, freedup, cdrkit, cdrecord, squashfs, ncdu, rsync, updatedb, dosfs). A Tabela 1 contém a listagem completa de todos os programas que compõem o repositório, sendo cada programa identificado por p_n, $1 \leq n \leq 80$.

4.2 Plataforma de experimentos

Todos os experimentos foram executados em um PC desktop x86 64 bits executando sistema operacional GNU/Linux (Gentoo) com kernel 3.11, composto por placa mãe Intel DG31PR, microprocessador Intel Core2 Quad de 2.66MHz, 2GB de memória RAM DDR2, disco rígido SATA com 5400 RPM e transferência nominal de 3.0Gb/s (pelo barramento SATA). Os códigos binários foram compilados e linkados dinamicamente com o biblioteca C GNU libc versão 2.2.5.

[1]Disponível em https://01.org/lkp

Tabela 1: Programas do repositório.

	CPU-Intensive		IO-Intensive
p_1	adpcm_code	p_{41}	Bonnie
p_2	adpcm_deco	p_{42}	badblocks
p_3	arithoh	p_{43}	base64
p_4	autcor—-	p_{44}	cat
p_5	bisection	p_{45}	cp
p_6	bubble_sor	p_{46}	cpio
p_7	bzip2	p_{47}	dd
p_8	combsort–	p_{48}	dirload_io
p_9	dhry2	p_{49}	disk
p_{10}	dhry2reg	p_{50}	du
p_{11}	dotprod—	p_{51}	duff
p_{12}	double	p_{52}	dupedit
p_{13}	dsp_mul32_	p_{53}	ffsb
p_{14}	fdct——	p_{54}	find
p_{15}	fft——-	p_{55}	fio
p_{16}	fibonacci-	p_{56}	freedup
p_{17}	float	p_{57}	fsck.fat
p_{18}	fractal—	p_{58}	genisoimage
p_{19}	gauss——	p_{59}	genload_io
p_{20}	genload_wh	p_{60}	io_thrash
p_{21}	goraud—-	p_{61}	ioping
p_{22}	gzip	p_{62}	iozone
p_{23}	hanoi	p_{63}	make-many-files
p_{24}	heap——	p_{64}	md5sum
p_{25}	insertion-	p_{65}	mkisofs
p_{26}	long	p_{66}	mksquashfs
p_{27}	mandelbulber	p_{67}	ncdu
p_{28}	max——-	p_{68}	od
p_{29}	mmq	p_{69}	randrw
p_{30}	pop_cnt—	p_{70}	recarray
p_{31}	primegen	p_{71}	rsync
p_{32}	radixsort-	p_{72}	seeker
p_{33}	redblack	p_{73}	shred
p_{34}	register	p_{74}	spew
p_{35}	selection-	p_{75}	split
p_{36}	short	p_{76}	sysbench
p_{37}	simpson13	p_{77}	tac
p_{38}	sobel——	p_{78}	tiotest
p_{39}	summillion	p_{79}	unsquashfs
p_{40}	vecsum—-	p_{80}	updatedb

4.3 Verificação do perfil das aplicações

Mesmo com o conhecimento prévio sobre o perfil de consumo dos programas do repositório, experimentos foram executados para analisar o perfil de consumo de recursos de cada programa diretamente na plataforma adotada para a execução dos experimentos, traçando suas características de consumo de CPU e E/S. Os dados de consumo em tempo de execução foram obtidos a partir do próprio monitor do kernel Linux, através dos arquivos especais */proc/stat*, */proc/statm* e */proc/io*. As métricas analisadas foram: tempo de execução em espaço de usuário (fração de tempo em que a CPU ficou ocupada processando instruções da aplicação), tempo de execução em espaço de kernel (fração de tempo em que a CPU ficou ocupada executando uma chamada ao sistema no kernel feita pela aplicação); tamanho do segmento de dados e da pilha; número de bytes lidos ou escritos em operações de E/S.

Para automatizar os procedimentos experimentais o uti-

litário *mtool*[2] foi desenvolvido. A ferramenta inicia o processo sob teste e monitora sua execução em uma taxa de 1 aferição por segundo (para minimizar os efeitos do processo de aferição no sistema) durante 30 segundos, provendo 30 amostras de aferições. Para os programas escolhidos neste estudo, foi verificado experimentalmente que 30 amostras são suficientes para os cálculos estatísticos de desvio padrão e intervalo de confiança. Após os 30 segundos, o processo é finalizado através do envio do sinal de sistema SIGKILL. Um cuidado especial foi tomado para que as entradas fornecidas aos programas garantissem sua execução durante 30 segundos. Os programas *CPU-Intensive* que utilizam entradas já pré-programas em código foram levemente modificados para efetuar a execução de seus algoritmos em *loop* evitando seu término antes dos 30 segundos necessários para aferição.

4.4 Diferentes compiladores e otimizações

Para investigar a influência de diferentes compiladores e *flags* de otimização no processo de agrupamento, todos os programas do repositório foram compilados com os compiladores gcc 5.4.0, icc 15.0.6 e llvm 3.8.1 com as *flags* *-O1*, *-O2*, *-O3* e *-Os*. Assim, cada par (compilador, *flag*) representa um único repositório de programas no qual o DAMICORE foi aplicado. Para cada árvore filogenética gerada, foi calculada a distância d_{raiz} que representa a distância entre um elemento da árvore e a raiz. Logo, caso as árvores de cada nível de otimização sejam semelhantes, o desvio padrão entre as respectivas distâncias d_{raiz} será baixo, indicando a robustez do agrupamento provido pelo DAMICORE, independente dos níveis de otimização ou compiladores utilizados.

5. RESULTADOS

5.1 Resultados das aferições

Os resultados das aferições de consumo de cada programa através da avaliação com o utilitário *mtool* são exibidos nas Figuras 1 e 2. A Figura 1 contém a média de consumo de CPU (em %) durante o tempo de avaliação (30 segundos) para cada um dos 80 programas do repositório (representados por uma barra, começando em p_1 ao p_{80}). Já a Figura 2 contém a média de operações de E/S em Bytes/s em escala logarítmica para melhor visualização. É possível observar que os primeiros 40 programas do repositório possuem alta taxa de consumo de CPU (próximos ou iguais a 100%). Já os programas de p_{41} ao p_{80} possuem baixa média (menor que 20%) com algumas exceções. Este resultado é importante pois detectou programas escolhidos, que por mais que possuam um perfil de consumo majoritariamente de E/S, e portanto classificados como *IO-Intensive*, possuem também características de programas com alto consumo de CPU. As quatro exceções foram os programas p_{41}, p_{63}, p_{65} e p_{67}, que obtiveram médias de consumo acima de 60%.

Em relação ao consumo de E/S, observa-se na Figura 2 que os programas classificados como *IO-Intensive* (p_{41} a p_{80}) apresentam maior média de operações de E/S em relação aos primeiros 40 programas (caracterizados como *CPU-Intensive*) aproximadamente em ordens superiores a 10^6.

As aferições executadas confirmaram os perfis dos programas selecionados para comporem o repositório com a detecção de quatro exceções, consistindo-se de programas caracterizados como *IO-Intensive*, mas que também possuem grande consumo de CPU. É importante notar que não é necessário a remoção destes programas do repositório, mas esta informação deve ser levada em consideração durante a análise dos resultados providos pelo DAMICORE.

[2]Disponível em https://github.com/rene/mtool

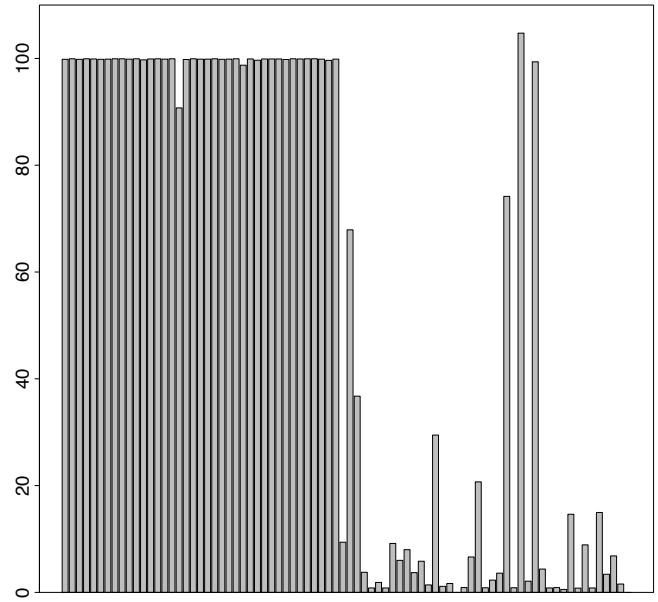

Figura 1: Média de consumo de CPU (em %) de cada programa do repositório.

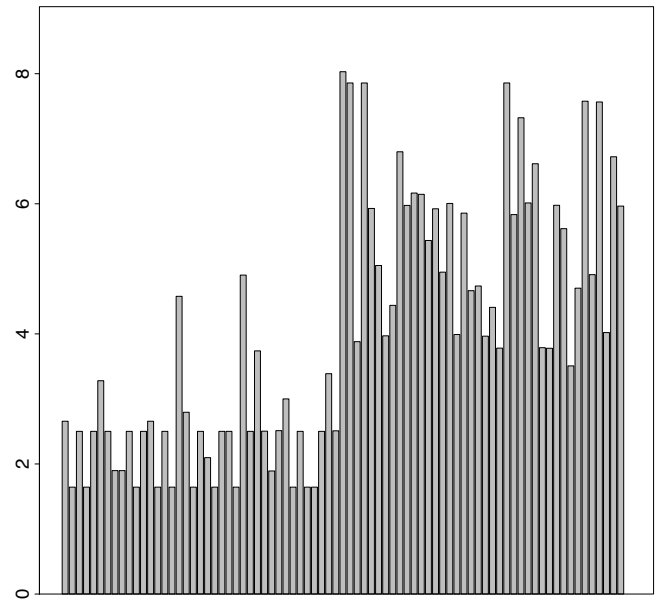

Figura 2: Média de operações de E/S em Bytes/s (log) de cada programa do repositório.

5.2 Resultados do DAMICORE

A Figura 3 contém a árvore filogenética resultante da aplicação do DAMICORE em uma amostra do repositório de 20 programas escolhidos aleatoriamente, sendo 10 *CPU-Intensive* e 10 *IO-Intensive*. A linha tracejada foi inserida manualmente para reforçar visualmente a divisão provida na árvore: todos os programas posicionados abaixo da linha tracejada são *IO-Intensive*, assim como todos os programas acima da linha são *CPU-Intensive*. É importante notar que a árvore determina a filogenia dos programas, ou seja, programas com comportamentos semelhantes (no que tange consumo de CPU ou E/S) foram agrupados próximos pois foi encontrada alguma semelhança entre os mesmos. Outras amostras também foram geradas aleatoriamente e passaram pela aplicação do DAMICORE produzindo resultados de agrupamentos semelhantes. O método Jacknife[3] também foi aplicado para fazer re-amostragem das amostras selecionadas não sendo detectadas mudanças nos agrupamentos nas árvores resultantes das novas amostragens.

A Figura 4 contém a árvore filogenética resultante da aplicação do DAMICORE sobre todo o repositório com os 80 programas. Novamente a linha tracejada foi inserida manualmente para reforçar visualmente a divisão entre os programas *CPU-Intensive* e *IO-Intensive* na árvore. Os nós sombreados contém os programas p_{41}, p_{63}, p_{65} e p_{67}, que não puderam ser caracterizados exclusivamente em uma das classes abordadas (*CPU* ou *IO-Intensive*).

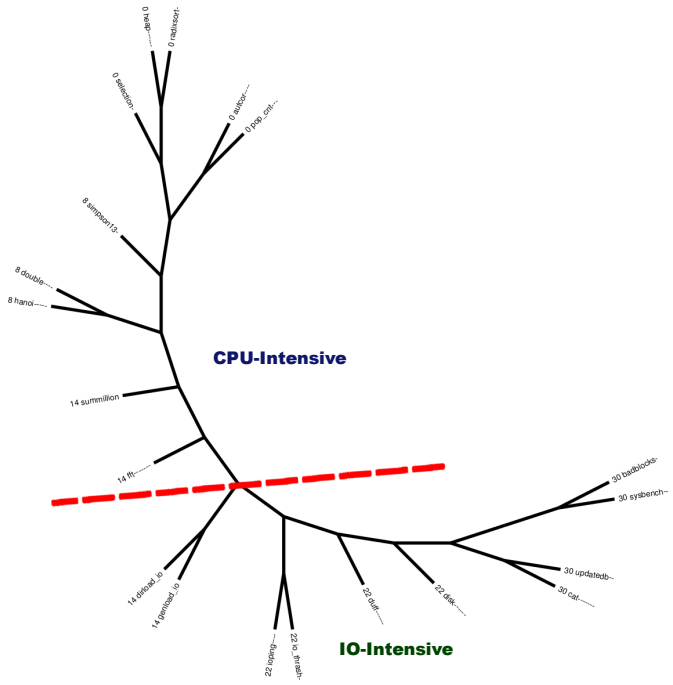

Figura 3: Árvore gerada para um subconjunto de 20 binários do repositório, sendo 10 *CPU-Intensive* e 10 *IO-Intensive*.

[3]Jacknife é um método estatístico que produz re-amostragem de uma determinada população retirando-se um elemento da mesma e avaliando o impacto nas métricas de estimação.

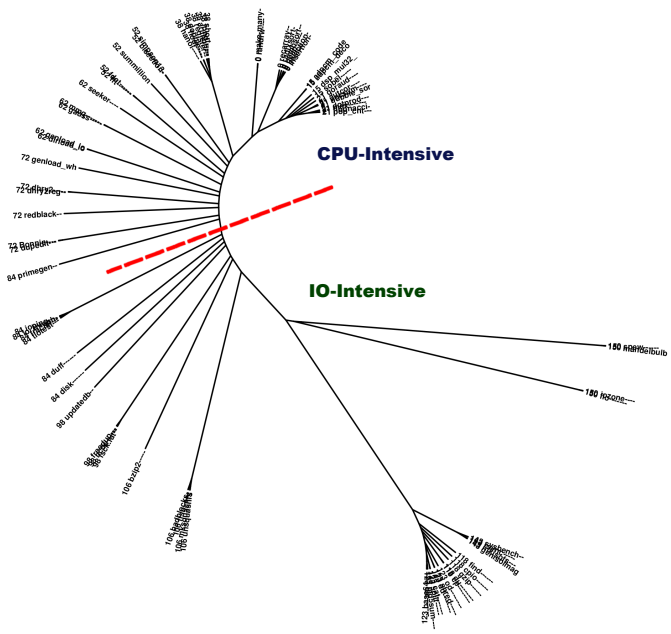

Figura 4: Árvore filogenética de todo o repositório de programas.

5.3 Estabilidade do método

As Figuras 5, 6 e 7 contém o desvio padrão da distância d_{raiz} considerando um determinado compilador e todas as *flags* de otimização *-O1*, *-O2*, *-O3* e *-Os* para cada um dos programas do repositório, cada um representado por uma barra no gráfico. Os resultados são mostrados para os compiladores *gcc*, *icc* e *llvm*, respectivamente.

Em todos os três compiladores o desvio padrão ficou abaixo de 0.04 (4%). O máximo desvio padrão obtido foi 0.038 com o programa p_{80} compilado pelo compilador *icc*.

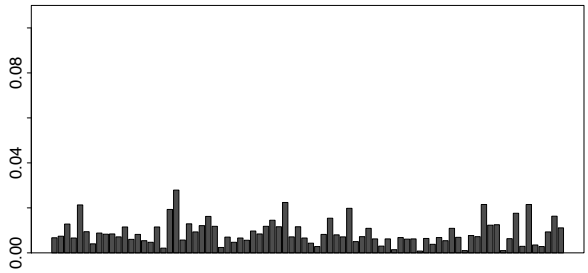

Figura 5: Desvio padrão das distâncias d_{raiz} para cada programa binário compilado com *gcc* e diferentes *flags* de otimização.

O cálculo da distância d_{raiz} e o respectivo desvio padrão baixo obtido mostra que as árvores geradas são semelhantes, mesmo quando compiladas com diferentes compiladores e níveis de otimização. Este resultado ressalta a estabilidade do método DAMICORE, que irá produzir resultados semelhantes para diferentes representações binárias do mesmo programa.

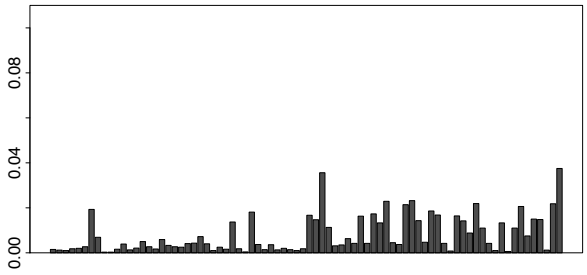

Figura 6: Desvio padrão das distâncias d_{raiz} para cada programa binário compilado com *icc* e diferentes *flags* de otimização.

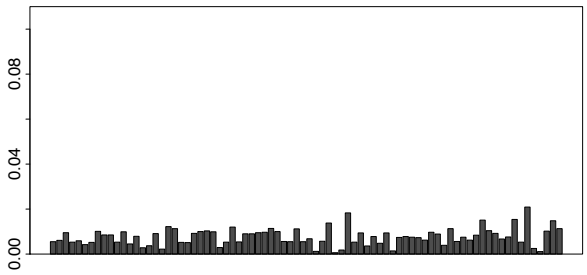

Figura 7: Desvio padrão das distâncias d_{raiz} para cada programa binário compilado com *llvm* e diferentes *flags* de otimização.

6. CONCLUSÕES

Programas executáveis binários podem ser considerados como uma representação em instruções de máquina de seus respectivos códigos fontes e algoritmos implementados. No contexto de consumo de recursos, os programas que apresentam comportamento semelhantes tenderão a utilizar instruções semelhantes. Portanto, a metodologia desenvolvida pela técnica DAMICORE pode ser utilizada dado que é agnóstica quanto à semântica dos dados analisados.

A técnica foi aplicada em um repositório de 80 programas que foram obtidos de diversas fontes entre trabalhos acadêmicos, utilitários de sistema ou *benchmarks* específicos. Analisando apenas os respectivos códigos binários (sem nenhum tratamento especial ou informação adicional fornecida) o DAMICORE foi capaz de produzir arvores filogenéticas que caracterizam os programas de acordo com seus respectivos perfis de consumo majoritariamente em CPU ou E/S. Esta informação pode ser utilizada para diversos fins, como escalonadores adaptativos, balanceadores de carga, provisionamento de QoS ou para comparação com outros programas.

Para avaliar o impacto do compilador e *flags* de compilação à aplicação da técnica, um estudo estatístico foi executado com três principais compiladores amplamente utilizados (*gcc*, *icc* e *llvm*). Os resultados mostraram que as árvores geradas pelo DAMICORE são consistentes e muito semelhantes (com desvio padrão máximo da ordem de 4%), o que mostra a estabilidade do método.

Trabalhos futuros incluem a avaliação de outros recursos consumidos do sistema, assim como a aplicação do DAMICORE para avaliação de outras características e comparação dos programas binários, podendo ser aplicado, por exemplo, na detecção de softwares maliciosos.

7. REFERENCES

[1] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pages 245–257. ACM, 2000.

[2] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi. Dynamically managing the communication-parallelism trade-off in future clustered processors. In *ACM SIGARCH Computer Architecture News*, volume 31, pages 275–287. ACM, 2003.

[3] A. S. Dhodapkar and J. E. Smith. Managing multi-configuration hardware via dynamic working set analysis. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pages 233–244. IEEE, 2002.

[4] J. Felsenstein and J. Felenstein. *Inferring phylogenies*, volume 2. Sinauer Associates Sunderland, 2004.

[5] E. J. FERREIRA, V. V. MELO, and A. C. B. DELBEM. Algoritmos de estimação de distribuição em mineração de dados: Diagnóstico do greening in citrus. In *II Escola Luso-Brasileira de Computação Evolutiva*, pages 1–1, Guimarães, Portugal, 2010.

[6] J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[7] P. D. Kaur and I. Chana. A resource elasticity framework for qos-aware execution of cloud applications. *Future Generation Computer Systems*, 37:14–25, 2014.

[8] H. Kim, H. Lim, J. Jeong, H. Jo, and J. Lee. Task-aware virtual machine scheduling for i/o performance. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 101–110. ACM, 2009.

[9] T. Lafage and A. Seznec. Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream. In *Workload characterization of emerging computer applications*, pages 145–163. Springer, 2001.

[10] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer Science & Business Media, 2013.

[11] K. Liu, H. B. K. Tan, and X. Chen. Binary code analysis. *Computer*, (8):60–68, 2013.

[12] H. A. Mahmoud, H. J. Moon, Y. Chi, H. Hacıgümüş, D. Agrawal, and A. El-Abbadi. Cloudoptimizer: multi-tenancy for i/o-bound olap workloads. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 77–88. ACM, 2013.

[13] L. G. MARTINS, R. NOBRE, A. C. Delbem, E. MARQUES, and J. M. CARDOSO. Exploration of compiler optimization sequences using clustering-based

selection. In *the 2014 SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems*, page 63, Edinburgh, 2014.

[14] L. G. A. Martins, R. Nobre, A. C. B. Delbem, E. Marques, and J. M. P. Cardoso. A clustering-based approach for exploring sequences of compiler optimizations. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 2436–2443, Beijing, 2014. doi: 10.1109/CEC.2014.6900634.

[15] R. Menotti. *LALP: uma linguagem para exploração do paralelismo de loops em computação reconfigurável*. PhD thesis, Universidade de São Paulo, São Carlos, maio 2010.

[16] L. F. Moro, C. L. Rodriguez, F. R. H. Andrade, A. C. B. Delbem, and S. ISOTANI. Caracterização de alunos em ambientes de ensino online. In *Workshop de Mineração de Dados em Ambientes Virtuais do Ensino/Aprendizagem, Anais do Congresso Brasileiro de Informática na Educação*, pages 1–10, Dourados, 2014.

[17] L. F. S. MORO, A. M. Z. LOPES, A. C. B. DELBEM, and S. ISOTANI. Osdesafios para minerar dados educacionais de forma rápida e intuitiva: o caso da damicore e a caracterização de alunos em ambientes de elearning.. in: Ii workshop de desafios da computação aplicada à educação. In *XXXIII Congresso da Sociedade Brasileira de Computação*, pages 1–10, Maceio, 2013.

[18] M. E. Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[19] A. Noureddine, S. Islam, and R. Bashroush. Jolinar: analysing the energy footprint of software applications. In *The International Symposium on Software Testing and Analysis*, pages Pages–445, 2016.

[20] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu. Understanding performance interference of i/o workload in virtualized cloud environments. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 51–58. IEEE, 2010.

[21] A. Sanches, J. M. Cardoso, and A. C. Delbem. Identifying merge-beneficial software kernels for hardware implementation. In *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, pages 74–79. IEEE, 2011.

[22] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *ACM SIGARCH Computer Architecture News*, volume 30, pages 45–57. ACM, 2002.

[23] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *ACM SIGARCH Computer Architecture News*, volume 31, pages 336–349. ACM, 2003.

[24] B. A. SILVA, A. C. B. DELBEM, P. C. DENIZ, and V. BONATO. Runtime mapping and scheduling for energy efficiency in heterogeneous multi-core systems. In *International Conference on ReconfigurableComputing and FPGAs*, pages 1–6, Mayan Riviera, 2015.

[25] P. Wilshire. Real-time linux: Testing and evaluation. In *Real Time Linux Workshop, Orlando, FL*, page 13, 2000.