IntPy: Um Interceptador para Python

Leonardo Fiório
Universidade Federal Fluminense
Niterói, Rio de Janeiro
leonardofiorio@id.uff.br

Mariana Teixeira
Universidade Federal Fluminense
Niterói, Rio de Janeiro
teixeiramariana@id.uff.br

Max Fratane
Universidade Federal Fluminense
Niterói, Rio de Janeiro
mfratane@id.uff.br

ABSTRACT

Atualmente, muitos cientistas desenvolvem scripts para automatizar determinadas tarefas científicas, que podem demorar um tempo considerável para serem executadas. Nossa proposta consiste na criação de um interceptador para scripts em Python. O objetivo da utilização dele é reduzir o tempo de execução de scripts que são constituídos por funções que possuem um mesmo comportamento e resultado quando chamadas com uma mesma entrada em diferentes execuções.

CCS CONCEPTS

• **Theory of computation** → *Caching and paging algorithms*; • **Software and its engineering** → *Scripting languages*;

KEYWORDS

Caching, Scripts, Interceptor

ACM Reference Format:
Leonardo Fiório, Mariana Teixeira, and Max Fratane. 2018. IntPy: Um Interceptador para Python. In *Proceedings of eScience*. ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

Experimentos científicos têm sido executados cada vez mais com o auxílio de procedimentos computacionais, como por exemplo *scripts*, visando lidar com o constante aumento dos volumes de dados e manipulações que podem ser efetuadas por cientistas [5]. Tais experimentos podem ser robustos, o que pode implicar na criação de *scripts* complexos e que consequentemente demoram uma quantidade notável de tempo para serem executados.

A performance pode ser baixa em *scripts* que tenham, por exemplo, criação excessiva de objetos ou diversas instruções encadeadas, e a repetição desse processo pode ser desnecessariamente cara caso a estrutura de referência ou os conteúdos dos dados não forem alterados [10]. Uma abordagem para diminuir o tempo de uma iteração seria alterar o código para salvar resultados intermediários de determinados estágios. Entretanto, essa abordagem consome um tempo considerável do programador, além de ser propensa a inserir novos bugs ao script [2].

O IntPy visa diminuir o tempo de iterações de *scripts* em *Python* armazenando em cache os valores de funções determinísticas e puras – que são funções que nunca alteram os valores e o estado do programa que realizou a chamada para essas determinadas funções [8] – anotadas pelo criador do script. Ele foi desenvolvido de forma a interceptar *scripts*, portanto ele será adicionado de forma transparente e chamado automaticamente [9] quando uma função estiver devidamente anotada, e será descrito de forma mais detalhada nas próximas seções.

eScience, June 2018, Niterói, RJ, Brasil

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 REFERENCIAL TEÓRICO

2.1 Hash

Uma função hash é uma função computacionalmente eficiente que mapeia cadeias binárias de comprimento arbitrário para cadeias binárias de comprimento fixo. A idéia básica de funções hash criptográficas é que um valor de hash serve como uma imagem representativa compacta de um valor de entrada e pode ser usado como se identificasse exclusivamente esse valor. Entretanto, a existência de colisões é garantida em mapeamentos muitos para um. A associação exclusiva entre entradas e valores de hash pode, na melhor das hipóteses, estar no sentido computacional. Um valor de hash deve ser exclusivamente identificável com uma única entrada na prática, e as colisões devem ser computacionalmente difíceis de ocorrer [3].

2.2 *Abstract Syntax Tree*

Abstract Syntax Tree (AST) ou Árvore de Sintaxe Abstrata é uma representação abstrata em forma de árvore onde cada nó denota uma construção que ocorre no código escrito em uma linguagem de programação, [6]. Através desta árvore se faz uma possível análise sintática e semântica do código, e por isso ela é muito utilizada por compiladores. A transformação do código em uma estrutura de dados hierárquica encadeada permite análise e tratamento do código de acordo com suas as restrições de escopo. Na figura 2 podemos visualizar uma AST construída a partir de um trecho de código apresentado na figura 1.

```
 9  if c = 15:
10      resultado = 0
11
12  a = num + 3
```

Figure 1: Exemplo de trecho de código para gerar AST

3 INTPY

O IntPy é uma ferramenta proposta para diminuir o tempo de execução de *scripts*, por meio do cache de valores intermediários de funções puras anotadas nas execuções do código. Esse tipo de ferramenta se caracteriza como um importante facilitador para o cientista que necessita de manipulação eficiente de grande volume de dados. Dada a grande variedade de aplicações desse tipo de *software*,

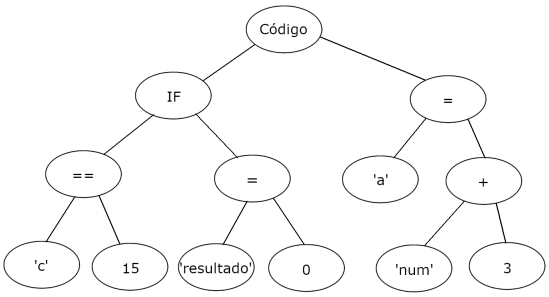

Figure 2: AST gerada a partir do trecho de código da Figura 1.

uma série de requisitos são desejáveis. Dentre as características, uma que se destaca é executar com o mínimo possível de restrições em relação à versão utilizada. Alguns programas que tem a mesma finalidade do IntPy apresentam restrições de versão do *Python* para sua perfeita execução [2]. É fácil observar que, nesta situação, o uso do software acaba se tornando prejudicado e muitas vezes pode ser descartado pelo cientista.

Por se tratar de uma ferramenta de otimização, o principal quesito a ser avaliado é o tempo de execução, que deve ser diminuído o máximo possível tanto na primeira execução do código, isto é, no momento em que a cache está sendo inicializada e preenchida com os valores resultantes, quanto nas execuções seguintes. Visando atender esse requisito, O IntPy utiliza hash para armazenamento e gerenciamento do cache, e através dele o IntPy facilmente consegue identificar alterações no código. Primeiramente, concatenamos nome da função, parâmetros, corpo da função e escopo de chamada da função, para depois aplicarmos a função de hash. Sendo assim, qualquer tipo de alteração, por menor que seja, culminará na geração de um código hash diferente. A partir disso, o IntPy consegue discernir se é possível utilizar o cache armazenado ou se deverá executar a função novamente devido a algum tipo de alteração efetuada pelo programador. Um obstáculo encontrado durante o processo de implementação diz respeito a restrição da utilização de caracteres especiais pelo sistema operacional em nomes de arquivos e diretórios. A utilização de funções hash nos conjuntos de dados, compostos pelo nome da função e seus parâmetros de entrada, possibilitou também a utilização do código hash gerado como nome dos arquivos que armazenam o conteúdo retornado. A função hash adotada foi a MD5. Esta função gera um código hash de 128 bits, que identifica um valor armazenado no cache e soluciona o problema de caracteres especiais.

A escolha de utilizar a AST se justifica pela possibilidade do IntPy realizar uma melhor análise das funções, suas estruturas, controles de variáveis, valores e escopos para que a aplicação dos valores armazenados em cache seja efetuado de forma correta. Durante o desenvolvimento deste projeto uma dificuldade foi encontrada: a identificação de diferentes tipos de modificações no código para verificar se o valor armazenado em cache de uma execução anterior ainda estaria válido. Por exemplo, alterações diretas nos parâmetros passados para as funções que o IntPy verifica são facilmente detectáveis, uma vez que o código hash gerado para cada execução é diferente. Porém, caso uma variável de escopo global utilizada pela função tenha seu valor alterado, ao manter os parâmetros inalterados, a abordagem que utiliza o hash da função não identificaria a mudança. Esse cenário se apresentou como um sério problema uma vez que um valor de cache errado poderia ser retornado pelo IntPy, que foi solucionado com o uso da AST.

3.1 Estrutura Gerada pelo IntPy

A ideia do IntPy é se manter uma ferramenta prática, visto que este tipo de característica implica em uma menor curva de aprendizagem. Com isso, o cientista que quiser utilizar o IntPy consequentemente dispende um menor tempo gasto no uso da ferramenta em si e maior tempo de dedicação do cientista na programação de sua aplicação. Sendo assim, sua utilização em um código consiste na importação da biblioteca implementada e a anotação "@deterministic" nas funções em que se deseja guardar os valores em cache. É importante observar que o programador, ao inserir tal anotação no código, deve garantir que a função é determinística. A Figura 3 é uma demonstração do uso do IntPy em uma função que calcula a série de Fibonacci, sabidamente determinística.

```
1  from src.intpy import deterministic
2
3  @deterministic
4  def fib(n):
5      if n < 2:
6          return n
7
8      return fib(n-1) + fib(n-2)
9
```

Figure 3: Exemplo de Fibonacci recursivo com a utilização do IntPy

A execução do script com o IntPy exibe um log que demonstra seu comportamento. As informações dele são importantes para verificação do momento em que foi criado o banco de dados, se a função executou e teve seu valor armazenado em cache, ou se não foi executada pois já havia o valor a ser retornado em cache.

Na primeira execução do IntPy, é criado um diretório oculto denominado ".intpy" no mesmo nível do programa que tem a anotação, que nesse exemplo é o script "fib.py", cuja estrutura é demonstrada na Figura 4. Este diretório armazena tudo que diz respeito ao cache do projeto em que o IntPy foi incorporado. Após essa etapa, o IntPy executa o script de criação do banco de dados SQLite e este gera um arquivo denominado "intpy.db". Com a estrutura de arquivos e o banco criado, considerando que na primeira execução o banco de dados e a estrutura estão vazios, o IntPy cria um novo arquivo do tipo ".ipcache" com nome sendo o MD5 do nome da função, parâmetros, corpo da função e escopo de chamada da função concatenados, na pasta ".cache" Esta decisão de projeto foi tomada para melhor desempenho visto que a abordagem com apenas a utilização do banco de dados implicaria em valores de tempo maiores. O banco de dados utilizado é o SQLite. Sua modelagem é bem simples e está demonstrada na Figura 5.

Considerando um cenário em que o script é chamado sem ser alterado e que, portanto, os valores de retorno das funções anotadas estão em cache, o fluxo do IntPy nesse caso consiste em gerar o MD5 dos valores relevantes (anteriormente citados) da função para realizar uma consulta ao banco de dados, que terá uma entrada

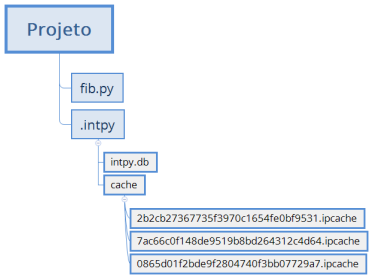

Figure 4: Estrutura de diretórios gerada pelas execuções do IntPy.

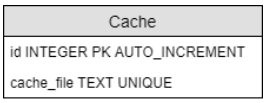

Figure 5: Estrutura do banco de dados criado pelo IntPy

relacionada a esse valor. Sendo assim, o IntPy resgata o arquivo com nome idêntico ao do MD5 gerado dentro da pasta ".cache", e faz a leitura do mesmo para devolver o valor da função anotada.

3.2 Exemplos de *scripts* com IntPy

Após a implementação, para fins de teste, alguns exemplos de funções foram implementadas e executadas sem o IntPy e com o IntPy para comparação de valores de tempo de execução. Em relação à função que calcula o Fibonacci, anteriormente demonstrada, o tempo de execução foi diminuído consideravelmente utilizando o IntPy. Ao executar a função para o cálculo do Fibonacci de 40 sem o IntPy o tempo de execução do algoritmo foi 49s. O tempo obtido na primeira execução do Fibonacci com o IntPy foi um pouco maior devido aos procedimentos internos intrínsecos do mesmo. No entanto, o resultado de uma nova execução do algoritmo para esse mesmo valor resultou no tempo 0.008s. Além disso, chamadas com valores maiores que 40 foram otimizadas, uma vez que só serão calculadas as chamadas recursivas em que o os valores não foram calculados anteriormente. Em uma nova execução do Fibonacci, passando 200 como parâmetro, o tempo de execução do algoritmo chegou a 25s. É fácil observar a diminuição do tempo de execução das funções com sua utilização.

Um outro exemplo interessante de aplicação do IntPy ocorre em casos de aplicação de programação dinâmica. Programação dinâmica é um método de elaboração de algoritmos em que resultados de subproblemas são reaplicados em novas execuções a fim de tornar possível resolução de determinado problema. Isso ocorre ainda no caso do Fibonacci. A execução de Fibonacci para 100 não é possível sem a utilização de programação dinâmica, porém, ao aplicar o IntPy, os valores em cache evitam a execução de todos os cálculos novamente possibilitando que se obtenha o resultado correto para números grandes.

Por outro lado, existem casos em que o IntPy não deve ser aplicado. Como explicado anteriormente, funções que não são determinísticas não devem ser utilizadas. Um simples exemplo de uma função que retorna um número inteiro aleatório dentro de um intervalo implica que a cada execução da função, mesmo que com as mesmas entradas, poderá retornar valores diferentes. Como consequência, neste caso não há como se aplicar o conceito de cache e, portanto, o IntPy não deve ser usado.

4 TRABALHOS RELACIONADOS

O IncPy [2] faz uma análise dinâmica que rastreia dependências, memoriza funções, gerencia o cache persistente, detecta acessibilidade de objetos, faz *profiling* de funções e detecta funções impuras. Apesar de não utilizar anotações, ele substitui o interpretador do *Python* 2.6.3 para determinar quais funções são determinísticas. Tal característica pode inviabilizar o uso para desenvolvedores que queiram utilizar funcionalidades do *Python* presentes em versões mais recentes.

Na literatura, existem diversos trabalhos relacionados ao armazenamento de valores já calculados para reutilização em aplicações industriais e/ou de larga escala.

O MemoizeIt [1] realiza uma análise dinâmica em programas Java que revela métodos que podem ser otimizados através de memoização e que sugere como implementar essas otimizações. Tal análise revela métodos em que a memoização pode ser benéfica porque um determinado método sempre gera a mesma saída dado um mesmo conjunto de entrada. O ponto principal do MemoizeIt é escalonar a abordagem de programas complexos orientados a objetos por meio de um algoritmo de análise iterativa que aumenta o grau de detalhamento da análise enquanto reduz o conjunto de métodos a serem analisados, entretanto para isso é necessário que o programa a ser analisado seja executado várias vezes. O desenvolvedor que utilizar o MemoizeIt, além decidir se deve e como implementar as sugestões de otimização, deve fornecer vários conjuntos de entrada para o programa, pois determinadas sugestões são dadas apenas para determinados conjuntos de entrada.

O Cachetor [7] é uma ferramenta de análise dinâmica em programas Java, que faz *profile* de programas visando ajudar desenvolvedores a encontrar oportunidades de caching para melhorar a performance. Ele possui três detectores para encontrar dados que podem entrar para cache, que fazem análise a nível de instrução, a nível de estrutura de dados, e a nível de chamada. Para que ele possa ser escalável em programas robustos, os autores desenvolveram uma técnica com abstrações em tempo de execução, que usa uma combinação de *profiling* de dependência dinâmica e *profiling* de valores para identificar operações que geram valores de dados idênticos. Tal ferramenta gera um alto *overhead* na execução das aplicações, entretanto é considerado aceitável dado que o foco da ferramenta é encontrar oportunidades de otimização. A diminuição desse *overhead* é considerada como trabalho futuro.

Xu [10] apresenta uma técnica dinâmica para encontrar estruturas de dados que podem ser reutilizadas, visando melhorar o desempenho de aplicações Java. Para expor ao desenvolvedor várias oportunidades de otimização, ele separa a capacidade de reutilização em três níveis: reutilização de instância, reutilização de estrutura e reutilização de dados, cada uma fornecendo uma perspectiva única para encontrar oportunidades de reutilização. Uma aproximação é efetuada para encontrar estruturas de dados que possam se encaixar nesses níveis. Entretanto, tal técnica não efetua o cache nem

reutiliza tais estruturas de dados durante a execução do programa. Tais funcionalidades são citadas como sendo trabalho futuro.

Um projeto focou melhorar desempenho através de padrões encontrados no código de *softwares* desenvolvidos. A abordagem adotada recebeu o nome de *Subsuming Methods Analysis* de sigla SMA [4]. Observando problemas de desempenho provocando aumento considerável do consumo de recursos de uma aplicação em produção está solução se apresentou eficiente conseguindo obter melhoras de até 50% no tempo gasto em execução e no processamento consumido. A ideia básica deste projeto foi identificar padrões de comportamento do software que possam ser otimizados, isso sem que haja um *overhead*.

Com isso, podemos perceber que tais abordagens podem consumir um tempo considerável do desenvolvedor por conta da análise dos resultados gerados por tais aplicações e reimplementação dos métodos para que os valores dos mesmos sejam cacheados ou para que eles executem de forma mais rápida.

5 CONSIDERAÇÕES FINAIS

O IntPy visa auxiliar de maneira prática cientistas diminuindo o tempo de execução de *scripts* em *Python* armazenando em cache os valores de funções puras anotadas pelo criador do script. Internamente, o IntPy utiliza hash para armazenamento e gerenciamento do cache e AST para realizar uma melhor análise das funções em vários aspectos, para que a aplicação dos valores armazenados em cache seja efetuado de forma correta. A utilização do IntPy em um código consiste na importação da biblioteca implementada e a anotação "@deterministic". Na primeira execução do IntPy é criado um diretório oculto na raiz do projeto, que armazena tudo que diz respeito ao cache do projeto em que o IntPy foi incorporado. A partir desse momento, a cada nova execução do script será verificada se há um valor de retorno já armazenado correspondente aos parâmetros passados, se houver o valor do cache é retornado, caso contrário o novo valor será armazenado em cache associado ao conjunto de valores dos parâmetros de entrada da função.

Durante os testes realizados foi possível identificar melhoras consideráveis de desempenho na execução de *scripts*, o que justifica seu uso para várias aplicações científicas. Porém, atualmente, o IntPy apresenta algumas limitações quanto ao seu uso. Sua implementação atual não consegue tratar funções aninhadas, ou seja, funções dentro de funções. Essa limitação é uma consequência de uma limitação da biblioteca Pickle utilizada pelo IntPy para o processo de serialização. Outra limitação na implementação ocorre em *scripts* com algum tipo de programação utilizando o conceito de *threads*. Um código que possui diferentes *threads*, isto é, fluxos de trabalhos diferentes entre processos, que utiliza as funcionalidades oferecidas pelo IntPy podem ter problemas na sua execução devido a possíveis instabilidades na concorrência de acesso ao cache.

Para versões futuras são previstas modificações na implementação para possibilitar sua utilização estável em *scripts* que possuam várias *threads*. O suporte a códigos com essa característica é muito importante, uma vez que é amplamente utilizado pela comunidade científica, especialmente para que se alcance um maior desempenho no processamento de dados. Outra modificação a ser efetuada para melhoria do IntPy consiste em possibilitar a utilização da cache em implementações com funções aninhadas, isto é, funções que estejam dentro de funções. Uma forma possível de desenvolver essa possibilidade é estudar outras formas de serialização para armazenamento de dados, talvez buscando alternativas diferentes do Pickle que deem esse tipo de suporte.

Por fim, podemos concluir que este tipo de aplicação pode ajudar muitos cientistas a facilitar o desenvolvimento e execução de várias aplicações. A ferramentada implementada, mesmo com ainda algumas limitações, já pode auxiliar vários trabalhos, como demonstrado em alguns exemplos testados durante o desenvolvimento.

REFERENCES

[1] Luca Della Toffola, Michael Pradel, and Thomas R Gross. 2015. Performance problems you can fix: A dynamic analysis of memoization opportunities. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 607–622.

[2] Philip J Guo and Dawson Engler. 2011. Using automatic persistent memoization to facilitate data analysis scripting. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. ACM, 287–297.

[3] Jonathan Katz, Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. 1996. *Handbook of applied cryptography*. CRC press.

[4] David Maplesden, Karl von Randow, Ewan Tempero, John Hosking, and John Grundy. 2015. Performance analysis using subsuming methods: An industrial case study. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, Vol. 2. IEEE, 149–158.

[5] Marta Mattoso, Cláudia Werner, G Travassos, Vanessa Braganholo, Leonardo Murta, Eduardo Ogasawara, F Oliveira, and Wallace Martinho. 2009. Desafios no apoio à composição de experimentos científicos em larga escala. *Seminário Integrado de Software e Hardware, SEMISH* 9 (2009), 36.

[6] F.P. Miller, A.F. Vandome, and M.B. John. 2010. *Abstract Syntax Tree*. VDM Publishing. https://books.google.com.br/books?id=5ns7YgEACAAJ

[7] Khanh Nguyen and Guoqing Xu. 2013. Cachetor: Detecting cacheable data to remove bloat. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 268–278.

[8] Alexandru Sălcianu and Martin Rinard. 2005. Purity and side effect analysis for Java programs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 199–215.

[9] Douglas C Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. 2013. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*. Vol. 2. John Wiley & Sons.

[10] Guoqing Xu. 2012. Finding reusable data structures. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 1017–1034.