Gerador de código para JVM a partir de fluxos de integração seguindo padrões EIP

Aluno: a definir
Orientador: Ricardo Luis de Azevedo da Rocha
Patrocinador: Opus Software
Laboratório de Linguagens e Técnicas Adaptativas
Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo

Resumo

Este projeto tem por objetivo investigar o processo de desenvolvimento de software sob o ponto de vista de integração. Houve mudança significativa no desenvolvimento de hardware (HENNESSY; PATTERSON, 2017; BARBIER, 2016) de tal forma que implicou em alterações também no software. De tal forma que as aplicações tornam-se mais voltadas para paralelismo e concorrência com um conjunto de novos problemas (YADAVA, 2015; BAINOMUGISHA et al., 2013; MAIER; ROMPF; ODERSKY, 2010).

O projeto envolve o desenvolvimento de uma aplicação com base em um editor web de fluxos de integração empregando Apache Camel conjuntamente com um gerador do código correspondente em Java. Este gerador deverá ser extensível de forma incremental.

Espera-se ao final que outras linguagens possam ser também incorporadas, agregando outros modelos de computação (e programação) como Scala (HALLER; SOMMERS, 2012) e Kotlin (JEMEROV; ISAKOVA, 2017).

Palavras-Chave

Desenvolvimento de Software, Geração de Código, JVM, Enterprise Integration Patterns.

Introdução

Nos dias atuais, o design do processador tornou-se principalmente multi-core (HENNESSY; PATTERSON, 2017); portanto, o desenvolvimento de software tornou-se mais focado na programação paralela simultânea, que trouxe problemas à memória compartilhada, como *data races* (HENNESSY; PATTERSON, 2017; BARBIER, 2016).

SoCs (*System on a Chip*) futuros estão se tornando plataformas de computação heterogêneas multi-núcleos, mas a programação paralela para multi/muitos-núcleos é uma tarefa difícil. É difícil para os programadores e mais difícil ainda para os sistemas automáticos gerar um programa paralelo.

Além disso, a programação multi-tarefa exige experiência e um alto nível de habilidade (MCKENNEY, 2011; YADAVA, 2015; GIMéNEZ, 2016). Mesmo escrever programas que usam apenas dois núcleos de forma eficiente para problemas paralelos não-triviais é uma tarefa difícil. E considera-se como quase impossível usando mais (4, 8, ...) núcleos, para qualquer aplicação paralela não-trivial. Outro grande desafio, no que diz respeito à escalabilidade de desempenho e portabilidade, é que o código paralelo é muito específico para uma plataforma ou configuração.

Dessa forma, uma tendência atual no desenvolvimento de software é adotar uma linguagem de programação funcional e programação reativa (BAINOMUGISHA et al., 2013; MAIER; ROMPF; ODERSKY, 2010).

As linguagens de programação tradicionais continuaram mudando ao longo de sua vida útil, e Java, C# e outros atualmente oferecem alternativas ao seu núcleo imperativo, incluindo recursos funcionais (BAINOMUGISHA et al., 2013; KAMINA; AOTANI, 2018; SALVANESCHI; MEZINI, 2013). No entanto, eles não lidam adequadamente com o problema crucial: threads simultâneos e/ou paralelos operando na memória compartilhada.

Concorrência

O desenvolvimento de software sofreu uma mudança significativa devido à mudança do núcleo único para o núcleo múltiplo no design da arquitetura do processador. Para extrair mais velocidade, os desenvolvedores precisavam projetar seu aplicativo com paralelismo em mente (HALLER; SOMMERS, 2012).

Infelizmente, obter a mesma velocidade de processamento provou ser extremamente difícil, como mostra a Lei de Amdahl (GUSTAFSON, 1988). Os aplicativos baseados na Web voltaram sua atenção para a escalabilidade horizontal, com o aumento da hospedagem e da computação em nuvem. Assim, a natureza desses aplicativos se tornou multithread e distribuída (D'ANGELO; MARZOLLA, 2014).

Paralelizar um aplicativo de API significa acessar simultaneamente o banco de dados. Em um ambiente com um único banco de dados, isso significa que as corridas de dados ocorrerão quando as solicitações precisarem de acesso exclusivo ao mesmo recurso (ZHENG et al., 2018). A declaração acima demonstra um impacto significativo no desempenho, pois uma transação deve bloquear recursos. A otimização de operações quando o recurso está bloqueado é possível, mas não altera o gargalo de desempenho.

Como Hewitt (HEWITT, 2009) afirmou sabiamente, "O advento da concorrência maciça por meio da computação em nuvem cliente e arquiteturas de computadores com muitos núcleos despertou interesse no modelo de ator".

O modelo computacional amplamente utilizado para definir a arquitetura do computador e as linguagens de programação é a Máquina de Turing. O design do processador,

linguagens que variam de Algol a C#, define um modelo interno baseado na transformação de estado (HOARE; JIFENG, 1998; SOTTILE; MATTSON; RASMUSSEN, 2009). O modelo de ator escolheu outra direção desde o início (HEWITT, 1977; AGHA, 2004), a sua base é o estilo de passagem de mensagens assíncrona, sem efeitos colaterais, assim ele fornece uma maneira diferente de lidar com a simultaneidade. Além disso, no modelo de ator, a ordem seqüencial é um caso particular, derivado da computação simultânea, que algumas linguagens funcionais como Scala permitem (HALLER; ODERSKY, 2007).

Proposta

A arquitetura das aplicações distribuídas apresenta um conjunto vasto de problemas cuja solução demanda proposições distintas, em geral baseada em troca de mensagens. No livro sobre Patterns of Enterprise Application Architecture de M. Fowler (FOWLER, 2002), apresenta-se um conjunto amplo de padrões de arquitetura para a solução de problemas. Neles observa-se que o emprego da troca de mensagens assíncronas torna-se uma medida não apenas desejável, mas necessária.

Obsera-se claramente que o uso de mensagens assíncronas colabora fortemente para a solução dessas questões, embora tenha a contrapartida de acrescentar novos desafios como (HOHPE; WOOLF, 2004): modelo de programação complexo, questões ligadas à sincronia (em cenários síncronos ou sequenciais), questões ligadas ao desempenho, existência de suporte à plataforma.

Esses problemas afetam o desenvolvimento de projetos, e algumas das práticas foram compiladas em padrões. O padrões adotados em projetos que envolvem integração foram compilados no denominado Enterprise Integration Patterns (HOHPE; WOOLF, 2004), que descreve diversos problemas e suas resoluções.

Este projeto se propõe a utilizar padrões para desenvolvimento de projetos aplicado ao problema de fluxos de integração gerando código para JVM. Inicialmente o código gerado será em linguagem Java, porque se partirá de um arcabouço conhecido, o Apache Camel (IBSEN; ANSTEY, 2018), que tem Java como plataforma para geração. Posteriormente pretende-se incluir outras linguagens de paradigmas distintos como Scala que é funcional, por exemplo.

Objetivos

O objetivo principal deste projeto de TCC é projetar e desenvolver uma aplicação com base em um editor web de fluxos de integração empregando Apache Camel conjuntamente com um gerador do código correspondente em Java. O gerador deverá ser desenvolvido de maneira a ser incrementalmente modificado para agregar outras linguagens como Scala, ou Kotlin, por exemplo.

Como objetivos secundários define-se a aplicação em si, um editor de fluxos de integração e um gerador de código em Java (que pode ser modificado de forma incremental). Todo o software será desenvolvido em código aberto.

Metodologia

Conhecer o desafio à frente é o primeiro passo para a concretização da proposta, inicialmente o(s) aluno(s) deverão estudar e conhecer os padrões, o arcabouço a ser empregado e a linguagem-alvo.

Esse ponto de partida permitirá definir conjuntamente com o patrocinador a aplicação a ser desenvolvida e pode-se começar o projeto.

Para o gerador de código pode-se optar por empregar um padrão de descrição do código intermediário que permite geração em diversas plataformas distintas, e desta maneira atingir-se o objetivo de produzir um gerador de código modificável incrementalmente. O candidato em questão é a LLVM (LATTNER; ADVE, 2004), que tem um conjunto amplo de ferramentas permitindo otimizar o código gerado.

Cronograma

O projeto terá seu início a partir de janeiro de 2020 e suas principais atividades são:

1. Estudar o Enterprise Integration Patterns (HOHPE; WOOLF, 2004) (importante seguir também o Patterns of Enterprise Application Architecture (FOWLER, 2002)).
2. Estudar o arcabouço existente (IBSEN; ANSTEY, 2018), seja do ponto de vista conceitual em suas bases formais (estatística, geração de código) quanto do ponto vista empírico (promover a execução de experimentos simples sem alterar o framework existente).
3. Auxiliar na definição da aplicação, e desenvolvê-la.
4. Projetar e desenvolver o editor Web de fluxos de integração.
5. Projetar e desenvolver o gerador de código – verificar a possibilidade de aplicar LLVM (LATTNER; ADVE, 2004).

Conclusão

A proposta de pesquisa aqui formulada está fortemente vinculada às atividades de pesquisa do LTA. Espera-se com este projeto a obtenção de um conjunto de programas e

de aplicações devidamente avaliadas e testadas.

Referências

AGHA, G. An algebraic theory of actors and its application to a simple object-based language. In: *In Ole-Johan Dahl's Festschrift, volume 2635 of LNCS*. Heidelberg, Germany: Springer, 2004. p. 26–57. Citado na página 3.

BAINOMUGISHA, E. et al. A survey on reactive programming. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 45, n. 4, p. 52:1–52:34, ago. 2013. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/2501654.2501666>. Citado 2 vezes nas páginas 1 e 2.

BARBIER, F. *Reactive Internet Programming: State Chart XML in Action*. New York, NY, USA: Association for Computing Machinery and Morgan & Claypool, 2016. ISBN 978-1-97000-177-8. Citado na página 1.

D'ANGELO, G.; MARZOLLA, M. New trends in parallel and distributed simulation: from many-cores to cloud computing. *CoRR*, abs/1407.6470, 2014. Disponível em: <http://arxiv.org/abs/1407.6470>. Citado na página 2.

FOWLER, M. *Patterns of Enterprise Application Architecture*. USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321127420. Citado 2 vezes nas páginas 3 e 4.

GIMéNEZ, D. A practical parallel programming course based on problems of the spanish parallel programming contest. *Procedia Computer Science*, v. 80, p. 1978 – 1988, 2016. ISSN 1877-0509. International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050916310067>. Citado na página 2.

GUSTAFSON, J. L. Reevaluating amdahl's law. *Commun. ACM*, ACM, New York, NY, USA, v. 31, n. 5, p. 532–533, maio 1988. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/42411.42415>. Citado na página 2.

HALLER, P.; ODERSKY, M. Actors that unify threads and events. In: MURPHY, A. L.; VITEK, J. (Ed.). *Coordination Models and Languages*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 171–190. ISBN 978-3-540-72794-1. Citado na página 3.

HALLER, P.; SOMMERS, F. *Actors in Scala*. USA: Artima Incorporation, 2012. ISBN 0981531652, 9780981531656. Citado 2 vezes nas páginas 1 e 2.

HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture; A Quantitative Approach*. 6th. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017. ISBN 9780128119051. Citado na página 1.

HEWITT, C. Viewing control structures as patterns of passing messages. *Artif. Intell.*, Elsevier Science Publishers Ltd., Essex, UK, v. 8, n. 3, p. 323–364, jun. 1977. ISSN 0004-3702. Disponível em: <http://dx.doi.org/10.1016/0004-3702(77)90033-9>. Citado na página 3.

HEWITT, C. A historical perspective on developing foundations iInfo(TM) information systems: iConsult(TM) and iEntertain(TM) apps using iDescribers(TM) information integration for iOrgs(TM) information systems. *CoRR*, abs/0901.4934, 2009. Disponível em: <http://arxiv.org/abs/0901.4934>. Citado na página 2.

HOARE, C.; JIFENG, H. *Unifying Theories of Programming*. Prentice Hall, 1998. (Prentice Hall series in computer science). ISBN 9780134587615. Disponível em: <https://books.google.com.br/books?id=WpdQAAAAMAAJ>. Citado na página 3.

HOHPE, G.; WOOLF, B. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Prentice Hall, 2004. (The Addison-Wesley Signature Series). ISBN 9780321200686. Disponível em: <http://books.google.com.au/books?id=dH9zp14-1KYC>. Citado 2 vezes nas páginas 3 e 4.

IBSEN, C.; ANSTEY, J. *Camel in Action*. 2nd. ed. USA: Manning Publications Co., 2018. ISBN 1617292931. Citado 2 vezes nas páginas 3 e 4.

JEMEROV, D.; ISAKOVA, S. *Kotlin in Action*. Manning Publications, 2017. ISBN 9781617293290. Disponível em: <https://books.google.com.br/books?id=qtcIkAEACAAJ>. Citado na página 1.

KAMINA, T.; AOTANI, T. Harmonizing signals and events with a lightweight extension to java. *Programming Journal*, v. 2, p. 5, 2018. Citado na página 2.

LATTNER, C.; ADVE, V. LLVM: A compilation framework for lifelong program analysis & transformation. In: *Code Generation and Optimization (CGO)*. Palo Alto, California: [s.n.], 2004. v. 1, n. 1. ISBN 0-7695-2102-9. Citado na página 4.

MAIER, I.; ROMPF, T.; ODERSKY, M. Deprecating the observer pattern. In: . [s.n.], 2010. Disponível em: <http://infoscience.epfl.ch/record/176887>. Citado 2 vezes nas páginas 1 e 2.

MCKENNEY, P. E. Is parallel programming hard, and, if so, what can you do about it? 2011. Citado na página 2.

SALVANESCHI, G.; MEZINI, M. Towards reactive programming for object-oriented applications. *Trans. Aspect-Oriented Software Development*, v. 11, p. 227–261, 2013. Citado na página 2.

SOTTILE, M.; MATTSON, T. G.; RASMUSSEN, C. E. *Introduction to Concurrency in Programming Languages*. 1st. ed. [S.l.]: Chapman & Hall/CRC, 2009. ISBN 1420072137, 9781420072136. Citado na página 3.

YADAVA, R. S. Review of the parallel programming and its challenges on the multicore processors. *Asian Journal of Computer Science and Technology*, v. 4, n. 1, p. 8–13, 2015. Citado 2 vezes nas páginas 1 e 2.

ZHENG, L. et al. Towards concurrency race debugging: An integrated approach for constraint solving and dynamic slicing. In: *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. New York, NY, USA: ACM, 2018. (PACT '18), p. 26:1–26:13. ISBN 978-1-4503-5986-3. Disponível em: <http://doi.acm.org/10.1145/3243176.3243206>. Citado na página 2.