UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PLANO DE TRABALHO PARA DOUTORADO

MELHORANDO A UTILIZAÇÃO DE RECURSOS POR PROCESSOS SUPORTADOS POR AMBIENTES DE EXECUÇÃO

CADIDATO
DANIEL LACET DE FARIA FIREMAN

FRANCISCO VILAR BRASILEIRO
ORIENTADOR

CAMPINA GRANDE
AGOSTO - 2016

Conteúdo

1 Introdução

Nos últimos anos um novo modelo de negócio surgiu, mudando a forma como os usuários têm acesso à Tecnologia da Informação (TI). Nesse modelo, a TI é oferecida como um serviço que pode ser adquirido quando necessário e a partir de qualquer equipamento que esteja conectado à Internet. Abaixo apresentamos uma definição mais completa/formal desse modelo, chamado de nuvem computacional (*cloud computing*):

> "Computação em Nuvem, como o próprio nome sugere, engloba as chamadas nuvens, que são ambientes que possuem recursos (hardware, plataformas de desenvolvimento e/ou serviços) acessados virtualmente e de fácil utilização. Esses recursos, devido à virtualização, podem ser reconfigurados dinamicamente de modo a se ajustar a uma determinada variável, permitindo, assim, um uso otimizado dos recursos. Esses ambientes são, em geral, explorados através de um modelo pay-per-use." [15]

Para viabilizar a migração da infraestrutura computacional, os aplicativos e os dados são movidos para grandes centros de processamento de dados, mais conhecidos como *data centers*. Os si temas de software presentes nos *data centers* provêm aplicações na forma de serviços na Internet [10]. Dessa forma, cria-se uma camada conceitual – a nuvem – que abstrai a infraestrutura para execução através de uma interface padrão que disponibiliza uma vasta gama de serviços. Uma vez que o usuário consiga se conectar a Internet, ele possui todos os recursos a sua disposição, sugerindo um poder e uma capacidade infinitos [16].

Em suma, podemos destacar três principais aspectos que são novos na computação em nuvem, em relação aos modelos anteriores [6]:

- Ilusão da disponibilidade de recursos ilimitados: o conceito da nuvem sugere que o usuário tem em suas mãos um conjunto muito grande de recursos e serviços;
- Eliminação de um comprometimento e/ou antecedência na contratação dos serviços: uma empresa pode começar usando poucos recursos de hardware aumentar essa utilização com a necessidade, sem que haja um comprometimento anterior em relação a essa quantidade;
- Habilidade de pagar somente pelos recursos que são utilizados (*pay-per-use*): uma vez definida uma métrica para cobrança (i.e. processadores por hora) o usuário paga somente pelos recursos utilizados.

Para aproveitar as simplicidade de acesso a infraestrutura e escalabilidade (horizontal e vertical) oferecidas pela computação na nuvem, o processo e as ferramentas de desenvolvimento começaram a mudar também. Nesse trabalho, focaremos nos ambientes de execução (*Runtime environments*). Mais precisamente, no impacto que o aumento de complexidade e funcionalidade que estes ambientes vem experimento tem no desempenho dos sistemas que eles dão suporte. No restante desta seção iremos explicar melhor esses ambientes de execução, bem como caracterizar melhor o problema a ser estudados e os objetivos desse trabalho.

1.1 Modelos e Ambientes de Execução

Um ambiente de execução (*runtime environment* ou *runtime system*), tem como principal função implementar partes do modelo de execução. Já o modelo de execução especifica a forma como o fluxo descrito pelo programa é executado. Cada linguagem de programação tem um modelo de execução, o qual tem sua especificação e implementação feita em conjunto com a linguagem.

Modelos de execução também pode existir independentemente de linguagens de programação, cujos exemplos seriam a biblioteca POSIX Threads [8], e o modelo de programação Map-Reduce [9], utilizado no Hadoop [14]. Compiladores e interpretadores também participam da implementação do modelo de programação.

Frequentemente não há critérios claros para decidir qual o comportamento de idioma é considerado dentro do sistema de execução contra a qual o comportamento é "compilada". Como um exemplo simples temos o ambiente de execução da linguagem C [12]. Esse ambiente de execução composto por um conjunto particular de instruções que são inseridas pelo compilador no programa executável. Entre outras coisas, estas instruções gerenciam a pilha processador, criam espaço para as variáveis locais e copiam parâmetros da chamadas de funções para o topo da pilha. No outro extremo ambientes de execução podem prestar serviços de uma ou máquina virtual, que se escondem até mesmo o conjunto de instruções do processador. Esta é a abordagem seguida por muitas linguagens interpretadas, tais como AWK [4] e algumas linguagens como Java [7], que são compiladas em um código de representação intermediária, o qual é independente de arquitetura do computador que vai ser executado (bytecode).

Abaixo citamos outros exemplos dessas funcionalidades [5]:

- Coleta de "lixo": a gerência automática do armazenamento alocado dinamicamente é um dos aspectos mais importantes de diversos ambientes de execução.
- Fluxo de entrada / saída: em sistemas operacionais como o Unix, que não têm oferecem facilidades para lidar com fluxo buferizado de entrada/saída, o processo deve fornecer seu próprio; este pode ser tratado pelo ambiente de execução.
- Chamadas do sistema operacional chama: funcionalidades do sistema operacional necessárias para um programa podem ser convenientemente exportadas pelo ambiente de execução.
- Gerenciamento de Interrupções e eventos assíncronos: se a linguagem de programação tem um mecanismo para lidar com eventos assíncronos, ela conta com o ambiente de execução para sua implementação
- Implementação de primitivas em linguagem de montagem: pode ser inconveniente para o compilador gerar código para algumas funcionalidades da linguagem de programação. Estas funções podem ser implementadas como chamadas de tempo de execução a rotinas exportadas pelo ambientes de execução.

Devido a características como eficiência, facilidade de programação e portabilidade, linguagens com ambientes de execução cada rico em funcionalidades vem sendo muito utilizadas no contexto de computação na nuvem.

1.2 Computação sem servidor (Serverless Computing)

Mesmo utilizando as facilidades de acesso a infraestrutura e escalabilidade disponíveis, a execução de sistemas na nuvem ainda demanda preocupações como dimensionamento da frota de servidores, armazenamento, banco de dados e tantas outras tarefas acessórias. A computação sem servidor veio como uma resposta a esses problemas: e se toda essa responsabilidade ficasse a cargo do seu provedor de nuvem?

A computação sem servidor (*serverless computing*) permite implantar grandes aplicações através de funcionalidades (funções) disparadas por gatilhos de ação (i.e. clicar num botão, chamadas a APIs e etc). O usuário só paga pelo uso do processamento quando o gatilho é acionado e os preços calculados em janelas de tempo pré-definidas (por exemplo, AWS Lambda usa uma janela de 100ms de uso)[1].

2 Problema e Objetivos

Do ponto de vista dos provedores de serviços em nuvem, vemos que a computação sem servidor vem ganhando muita aceitação no mercado. Ao fazer uma pesquisa entre os principais provedores que oferecem de esse serviço [1, 2, 3] vemos que a grande maioria oferece suporte a escrita de funções em linguagens com ambientes de execução bastante complexos, por exemplo, Node.js, Java, C# e Python.

Do ponto de vista dos clientes, temos que as linguagem suportadas nativamente pelos principais provedores de serviços na nuvem também são tem seu funcionamento auxiliado por ambientes de execução bastante complexos.

Uma vez que custo de execução adicionado pelo ambiente de execução depende de aspectos tais como[13, 11]:

1. aplicação (ões) executando + carga a que a(s) aplicação(ões) está(ão) sujeita(s)
2. servidor (real ou virtual) utilizado

Temos que otimizar aspectos como i) as funções/aplicações que serão co-alocadas, ii) parâmetros de configuração do ambiente de execução e iii) do servidor podem ter um impacto muito grande no custo total de execução dos clientes e/ou provedores de serviços. Estes são os principais problemas que este trabalho se propõe a resolver.

Assim, o objetivo geral deste trabalho é projetar, desenvolver e avaliar soluções que torne mais eficaz a utilização de recursos por processos suportados por ambientes de execução.

Espera-se que as seguintes metas sejam alcançadas no decorrer do doutorado:

- Entender o funcionamento das diversas opções de ambientes de execução disponíveis. Isso inclui propor e validar um modelo que consiga explicar o impacto de ambientes de execução;
- Desenvolvimento de mecanismos de configuração e otimização de processos suportados por ambientes de execução baseado em características da aplicação, da carga e do ambiente de execução;
- Desenvolvimento de mecanismos de configuração e otimização de processos suportados por ambientes de execução baseado em características resultantes da co-alocação de aplicações, bem como da carga imposta a cada uma delas e do ambiente de execução;
- Implementação serviços de auxilio à configuração e otimização processos suportados por gerenciadores de execução que funcione para as principais linguagens de programação e ambientes de execução. O principal objetivo desse serviço é permitir a validação dos mecanismos propostos.

3 Metodologia de pesquisa e estratégia de ação

3.1 Metodologia

Esta seção apresenta um detalhamento da metodologia utilização para execução desse plano de trabalho. A metodologia de pesquisa será predominantemente empírica, porém este não será o único método. Dois métodos principais de pesquisa científica serão combinados: pesquisa sistemática e pesquisa quantitativa.

A pesquisa sistemática de temas relacionados servirá para identificar com clareza o estado-da-arte nas diversas áreas de pesquisa relacionadas com o tema em estudo.

A pesquisa empírica quantitativa tem como principal objetivo avaliar os benefícios dos mecanismos propostos em cenários reais, ou próximos do real. Pretende-se usar duas técnicas de avaliação: avaliação baseada em experimentos de simulação e avaliação baseada em experimentos de medição. Os simuladores serão utilizados para implementação de modelos e sua validação será feita com a realização de experimentos de medição executados em um ambiente real. Além disso, todo o software desenvolvido será verificado com testes de unidade, integração e testes de sistema, bem como testes de carga e de regressão.

3.2 Estratégia de ação para execução do plano

Visando atingir as metas propostas, o plano de trabalho será executado em três duas principais, cobrindo cada um dos temas discutidos anteriormente:

1. desenvolvimento de mecanismos para diminuição do impacto negativo da um *runtime* na utilização de CPU e memória RAM;
2. desenvolvimento de um serviço para auxilio à configuração de *runtimes* baseado nas características de execução das aplicações e da carga aplicada;
3. generalização/adaptação dos mecanismos citados acima para *runtimes* das linguagens mais utilizadas no cenário global;
4. adaptação do mecanismo para gerência semi-automática online (reconfiguração)

Cada uma dessas fases seguirá o mesmo conjunto de etapas descritas a seguir:

Etapa 1: revisão bibliográfica

Nesta etapa será realizada uma extensa revisão bibliográfica, com o intuito de identificar os trabalhos relacionados, as soluções propostas na literatura e possíveis ideias para a solução dos problemas específicos da fase correspondente.

Etapa 2: formalização do problema

Ao final desta etapa será produzida uma formalização do problema estudado e a especificação clara dos requisitos que uma solução para o mesmo deve atender.

Etapa 3: estudo e modelagem da proposta de solução

Esta etapa compreende a modelagem do sistema estudado, a partir da formalização realizada na etapa anterior. Ao final desta etapa, serão identificadas uma ou mais abordagens para solucionar os problemas estudados.

Etapa 4: avaliação da solução proposta escolhida

A eficiência das abordagens selecionadas será avaliada através da análise de modelos analíticos e/ou resultados de simulação. Esta etapa envolve o planejamento e a execução dos experimentos de simulação, a definição dos parâmetros de entrada e das métricas de desempenho que serão coletadas.

Etapa 5: implementação e validação

Nessa etapa as abordagens mais promissoras serão implementadas e o software gerado será usado para executar experimentos em um sistema real que servirão para validar os modelos analíticos e/ou os resultados de simulação.

Etapa 6: escrita de artigos e relatórios

Além das etapas descritas anteriormente, ao longo do desenvolvimento de todo o trabalho de pesquisa serão gerados artigos científicos e relatórios de progresso.

4 Cronograma de atividades

As tabelas a seguir indicam quando cada uma das etapas de cada uma das fases descritas anteriormente serão executadas.

Fase 1)

Ano	Trimestre	Etapa 1	Etapa 2	Etapa 3	Etapa 4	Etapa 5	Etapa 6
2016	4	X					
2017	1		X	X			
2017	2				X		
2017	3					X	
2017	4						X

Fase 2)

Ano	Trimestre	Etapa 1	Etapa 2	Etapa 3	Etapa 4	Etapa 5	Etapa 6
2018	1	X					
2018	2		X	X			
2018	3				X		
2018	4					X	X

Fase 3)

Ano	Trimestre	Etapa 1	Etapa 2	Etapa 3	Etapa 4	Etapa 5	Etapa 6
2019	1	X	X				
2019	2			X			
2019	3				X		
2019	4					X	X

Fase 4)

Ano	Trimestre	Etapa 1	Etapa 2	Etapa 3	Etapa 4	Etapa 5	Etapa 6
2020	1	X	X				
2020	2			X	X		
2020	3					X	X

Referências

[1] AWS Lambda. https://aws.amazon.com/lambda/details/. Último acesso: 2016-08-02.

[2] Google Clould Functions. https://cloud.google.com/functions/. Último acesso: 2016-08-02.

[3] MS Azure Functions. https://azure.microsoft.com/en-us/services/functions/. Último acesso: 2016-08-02.

[4] A. V. Aho, B. W. Kernighan, and P. J. Weinberger. *The AWK Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.

[5] A. W. Appel. A runtime system. *LISP and Symbolic Computation*, 3(4):343–380, 1990.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[7] K. Arnold, J. Gosling, and D. Holmes. *The Java Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 2000.

[8] D. R. Butenhof. *Programming with POSIX Threads*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[10] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali. Cloud computing: Distributed internet computing for it and scientific research. *IEEE Internet Computing*, 13(5):10–13, Setembro 2009.

[11] S. Jayasena, M. Fernando, T. Rusira, C. Perera, and C. Philips. Auto-tuning the java virtual machine. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop, IPDPS 2015, Hyderabad, India, May 25-29, 2015*, pages 1261–1270, 2015.

[12] B. W. Kernighan. *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd edition, 1988.

[13] C.-T. D. Lo, W. Srisa-an, and J. M. Chang. A performance comparison between stop-the-world and multithreaded concurrent generational garbage collection for java. In *Performance, Computing, and Communications Conference, 2002. 21st IEEE International*, pages 301–308, 2002.

[14] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[15] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, Dezembro 2008.

[16] J. Voas and J. Zhang. Cloud computing: New wine or just a new bottle? *IT Professional*, 11(2):15–17, Março 2009.