Suporte às atividades de manutenção de software em bases de dados abertas e distribuídas

Nathan Manera Magalhães, Marco A. P. Araújo, José Maria N. David, Laércio Pioli, Mário A. R. Dantas

Programa de Pós-Graduação em Ciência da Computação (PPGCC) – Universidade Federal de Juiz de Fora (UFJF), Juiz de Fora – MG – Brasil

nathan.manera@gmail.com, marco.araujo@ice.ufjf.br, jose.david@ufjf.edu.br, laerciopioli@ice.ufjf.br, mario.dantas@ice.ufjf.br

Abstract. *Identifying globally distributed software developers, specialists in a specific technology, for the software maintenance has become a complex activity, with high time consumption and susceptible to decision failures when carried out in a single local database. This increasing complexity is mainly due to the requirements that contemporary software systems demand. In this context, the search for specialists for maintenance activities on diverse databases is necessary, which requires an increasing processing power to retrieve information in distributed databases. This paper presents an approach that aims to provide an environment that deals with the search for specialists in different repositories to support decision making for software maintenance. Our proposal was executed in a conventional computational environment and in a high performance one, the latter presenting a differential in relation to the distributed databases' processing.*

Resumo. *A identificação de desenvolvedores de software globalmente distribuídos, especialistas em uma tecnologia, para a manutenção de software tornou-se uma atividade complexa, com alto consumo de tempo e suscetível a falhas de decisão quando realizada em uma única base de dados local. Esta crescente complexidade se deve, sobretudo, aos requisitos que os sistemas contemporâneos demandam. Neste contexto, a busca por especialistas para as atividades de manutenção em bases diversificadas tornou-se necessária, o que exige um poder de processamento crescente para recuperar as informações em diferentes repositórios. Este artigo apresenta uma abordagem que visa prover um ambiente que trata a busca de especialistas em diferentes repositórios com o objetivo de apoiar a tomada de decisão para a manutenção de software. Nossa proposta foi executada em um ambiente computacional convencional e em um de alto desempenho, este último apresentando um diferencial em relação ao processamento das bases de dados distribuídas.*

1. Introdução

Atividades de manutenção são essenciais para manter o pleno funcionamento de um software durante seu ciclo de vida. Como exemplos tem-se a adição de novas funcionalidades no software com tarefas do tipo, adaptativa e evolutiva tais quais requisições de mudança (*change request*), ou então por meio de tarefas do tipo corretiva para correção de defeitos expostos pelos relatórios de erros (*bug report*). A fase de manutenção compõe a maior parte do ciclo de vida de um software. Segundo Erlikh

(2000), esta fase despende cerca de 90% de todo o esforço do desenvolvimento de um software.

Com o passar do tempo, os projetos de software aumentaram em escala, tamanho e complexidade diante da variedade de tecnologias que surgiram nos últimos anos. A manutenção de software tornou-se desafiadora devido ao crescente número de problemas (*bugs*) relatados em programas de software complexos e de grande escala (Goyal e Sardana, 2017).

Escolher desenvolvedores apropriados para cada tarefa de manutenção tornou-se então uma atividade complexa, com alto consumo de tempo e suscetível a falhas de decisão quando realizada manualmente. Esta complexidade se deve, sobretudo, à variedade de *expertises* necessárias e aos diferentes repositórios que podem permitir a localização de especialistas nas diferentes áreas tratadas pelo software. À medida que a complexidade do software aumenta a necessidade de encontrar especialistas em diferentes repositórios, geograficamente distribuídos também aumenta. Como resultado, necessitamos de um suporte computacional para apoiar as atividades de manutenção. Adicionalmente, é necessário minimizar as atribuições errôneas de desenvolvedores desprovidos dos conhecimentos exigidos pelas tarefas. Pois a inaptidão de um desenvolvedor pode ocasionar em tentativa fracassada deste em executar uma nova contribuição em manutenção de software, e com isso, resultar em tempo perdido e altos custos de manutenção. Esse despreparo pode estar associado à falta de conhecimentos necessários para a tarefa (reputação baixa em *expertises*); ou à inaptidão com os conhecimentos exigidos para uma tarefa devido ao desuso destes conhecimentos (esquecimento).

Soluções para apoiar a manutenção de software têm sido propostas frequentemente na literatura. Miguel et al. (2016) e Lélis et al. (2016) ao abordarem o apoio à manutenção de software propuseram, respectivamente, soluções que preveem o esforço de tempo a ser gasto em uma nova tarefa e que enfocam na manutenção colaborativa associada à reputação dos desenvolvedores geograficamente distribuídos. Esses trabalhos fornecem suporte para o apoio à tomada de decisão na escolha de um desenvolvedor apto a uma tarefa. Porém, os autores não tratam da aplicação de suas propostas em bases abertas (*open source*) e não consideram o suporte à utilização de diferentes bases para tratar a complexidade do software. Ao utilizarmos diferentes bases é possível acessar uma diversidade de projetistas e desenvolvedores, geograficamente distribuídos, com diferentes *expertises* capazes de apoiar as atividades de manutenção. Adicionalmente, não consideram o esquecimento relacionado ao conhecimento dos desenvolvedores.

Este trabalho tem como objetivo tratar a indicação de desenvolvedores devidamente preparados para cada tarefa de manutenção. O trabalho propõe tratar a reputação e o esquecimento no contexto de diferentes repositórios abertos (*open source*). Como resultado, buscamos minimizar o esforço, e tempo, despendidos com a busca por desenvolvedores devidamente capacitados para tarefas de correção e aperfeiçoamento de software. Entretanto, para o processamento de uma diversidade de bases de dados abertas, geograficamente distribuídas, necessitamos de um suporte computacional capaz de apoiar o processamento dessas bases, considerando a *performance* e a escalabilidade.

A identificação dos perfis dos desenvolvedores ocorre com base nos conhecimentos aplicados por estes nas tarefas que lhes foram anteriormente atribuídas e finalizadas com sucesso. É também intento da abordagem considerar contribuições externas em plataformas de desenvolvimento de software para enriquecer o perfil com informações de *expertise*. Ao identificar os desenvolvedores devidamente capacitados, estes podem ser apresentados como indicados para uma tarefa que exija seus conhecimentos. É também proposta do trabalho sugerir tarefas relevantes para desenvolvedores aptos. A decisão final de escolha cabe ao responsável pelas atribuições e ao próprio desenvolvedor em atender às sugestões que lhes são apresentadas.

Este artigo está organizado em 6 seções, incluindo a Introdução. A seção 2 discorre sobre a Fundamentação Teórica. A seção 3 discute os trabalhos relacionados. A seção 4 disserta sobre a proposta desenvolvida para a indicação de desenvolvedores em tarefas de manutenção de software. A seção 5 tem por objetivo apresentar a avaliação proposta em um ambiente de alto desempenho. E, por fim, a seção 6 apresenta uma síntese com as contribuições do trabalho e perspectivas futuras.

2. Fundamentação Teórica

Um passo importante na escolha de um desenvolvedor para uma tarefa é o estabelecimento de confiança deste com o recrutador da tarefa. Confiança esta que pode ser estabelecida mediante reputação do desenvolvedor sobre as *expertises* utilizadas na manutenção de software. A confiança se relaciona também com o fator esquecimento. Para ser confiável espera-se que o desenvolvedor esteja atualizado em relação ao conhecimento exigido para a realização da tarefa de manutenção.

2.1. *Crowdsourcing*

Quando não há desenvolvedores locais disponíveis para serem recrutados, opta-se pela adoção do desenvolvimento global de software, em que participam desenvolvedores geograficamente distribuídos, com diversas especialidades. Esse modelo de negócio conhecido como *crowdsourcing* (De Neira et al. 2018), busca fazer com que programadores engajados na busca por trabalho se conectem com tarefas de desenvolvimento.

Um desenvolvedor consegue se destacar como notório nas suas *expertises* de atuação quando ele apresenta boa reputação na comunidade de *crowdsourcing*. Reputação essa que é construída por meio das boas contribuições apresentadas ao longo da carreira.

2.2. Esquecimento

Em uma abordagem que busca identificar *expertises* de um desenvolvedor com base em suas contribuições passadas, um fator importante a ser considerado é a não utilização do conhecimento por algum período de tempo. A esse fator denomina-se esquecimento ou obsolescência do conhecimento. Khatun e Sakib (2016) observam que quanto mais recente um conhecimento tenha sido utilizado, mais apto o desenvolvedor estará na tarefa da respectiva *expertise*, indicando assim maior familiaridade com o conhecimento.

Em abordagens que recomendam desenvolvedores com base em suas contribuições históricas, considerar o fato da *expertise* ociosa é fundamental para

efetuar indicações mais condizentes com o atual estado de conhecimento do desenvolvedor. Havendo novas interações, o desenvolvedor recupera a familiaridade com o código, e então os valores obtidos divergem dos valores previstos na curva de esquecimento.

Hattori et al. (2012), consideram o fator esquecimento utilizando a familiaridade de desenvolvedores com cada artefato de um sistema de software para determinar quem será o proprietário do código (*code ownership*). Este conceito informa a quantidade de conhecimento acumulado por cada desenvolvedor nos artefatos de um sistema de software, de modo a identificar os mais familiarizados com cada artefato.

As soluções propostas na literatura tratam o fator esquecimento ao encontrar desenvolvedores em bases de software. Entretanto, elas não relacionam o fator esquecimento com a reputação de diferentes desenvolvedores em diferentes bases de dados geograficamente distribuídas, nem tampouco oferecem recursos para apoio às decisões sobre o especialista mais adequado à atividade de manutenção.

3. Trabalhos Relacionados

Encontrar especialistas para apoiar o desenvolvimento de software em um contexto global é uma atividade que tem sido tratada na literatura. Entretanto, esses trabalhos apresentam limitações, principalmente em relação à diversidade e à escalabilidade de repositórios capazes de apoiar a complexidade das aplicações. Esta diversidade de repositórios acaba gerando uma volatilidade de dados que devem ser tratados pelas aplicações.

Miguel et al. (2016), por exemplo, elaboraram o *framework*, que utiliza os dados históricos dos projetos de software para apoio à atividade de estimativa de esforço de manutenção, visando calcular o tempo e planejamento das manutenções. É verificada também a reputação de um desenvolvedor com base na quantidade de requisições que este resolveu ao longo do tempo. Apesar de o *framework* oferecer suporte para planejar as atividades de manutenção, a tarefa de alocação dos desenvolvedores às atividades de manutenção não é abordada e o trabalho não considera o contexto de diferentes bases abertas (*open source*) de desenvolvimento, nem tampouco o fator de esquecimento associado ao especialista.

Trainer et al. (2018) apresentam um trabalho que trata sobre a influência de fatores da confiabilidade por percepção de suas entidades. Para tanto, propuseram uma ferramenta de apoio à identificação de desenvolvedores confiáveis aos seus recrutadores. A respectiva abordagem coleta informações sobre as atividades de desenvolvimento de software em repositórios de códigos-fonte e em sistemas de rastreio de *bugs*. A alta disponibilidade e resposta rápida do desenvolvedor atuando em uma *expertise* são indicadores do seu conhecimento. Entretanto, os autores não consideram que os indicadores do conhecimento sobre um tema específico e a reputação do desenvolvedor podem ser obtidos de diferentes repositórios abertos.

De Neira et al. (2018) elaboraram um estudo para identificação de "hiperespecialistas" em uma plataforma de desenvolvimento de software por *crowdsourcing*. Nessa plataforma, denominada *TopCoder*[1], são disponibilizadas tarefas

[1] https://www.topcoder.com/

de manutenção e/ou desenvolvimento de software para desenvolvedores globalmente distribuídos resolverem em modalidade de competição. Os autores observaram que os desenvolvedores dessa plataforma apresentam um comportamento para resolverem tarefas de tecnologias específicas. No entanto, o estudo não considerou o esquecimento ao fazer a avaliação das contribuições passadas em diferentes repositórios *open source*.

Oliveira Jr et al. (2019) elaboraram uma arquitetura de apoio à recomendação de colaboradores advindos de plataformas externas para atuarem em projetos de desenvolvimento global de software. A abordagem faz uso de informação semântica proveniente de uma combinação de ontologias para buscar contribuições passadas dos desenvolvedores. Porém, no respectivo trabalho, não é considerado o fator esquecimento na atribuição das *expertises* (conhecimentos) nos desenvolvedores, algo que pode ser um problema ao se recomendar alguém inativo.

Todas estas abordagens mencionadas se relacionam no sentido de estabelecer uma forma de confiança entre desenvolvedores através de fatores como, reputação, perfil de *expertise*, estimativas de tempo despendido, inferência de *expertises* externas. Porém, não tratam da complexidade das buscas em diferentes repositórios abertos, globalmente distribuídos, nem tampouco o fator esquecimento capazes de apoiar a decisão em relação aos especialistas para uma determinada tarefa. Dada a complexidade do software, a busca pelo conhecimento em relação aos especialistas globalmente distribuídos poderá apoiar a realização de atividades de manutenção do sistema. Entretanto, soluções que tratam do volume de dados envolvidos necessitam ser utilizadas.

4. Solução Proposta

Com o objetivo de apoiar a seleção de especialistas para as atividades de manutenção, uma arquitetura foi desenvolvida considerando-se o suporte de repositórios distribuídos e de um suporte computacional de alta *performance* visando a escalabilidade da solução. Neste contexto, um sistema foi especificado de acordo com os requisitos, a seguir.

Foram definidos os requisitos funcionais e não funcionais para a solução proposta. Como requisitos funcionais, cabe citar que o sistema: (i) deve permitir apoio à tomada de decisão de um desenvolvedor responsável por atribuir usuários para tarefas de manutenção de software; (ii) deve ser capaz de extrair os conhecimentos aplicados em contribuições externas do usuário advindas de repositórios *open source*, tais como: *GitHub*, *TopCoder* e *StackOverflow*, entre outros; (iii) deve ser capaz de utilizar os critérios de reputação sobre os dados das contribuições externas e internas do usuário; (iv) deve apresentar uma lista de usuários relevantes para cada tarefa nova (não atribuída); (v) deve apresentar uma lista de sugestão de tarefas relevantes para cada usuário; (vi) deve aplicar uma taxa de decaimento do conhecimento (esquecimento) conforme o tempo de desuso das *expertises* associadas ao usuário; e (vii) deve prover apoio à formação de grupos de usuários relevantes a uma tarefa como alternativa à atribuição de apenas um usuário.

Como requisitos não funcionais associados, cabe mencionar: (i) **Escalabilidade**: permitir a utilização de diferentes bases de dados (*open source*) em função das diferentes complexidades do software; (ii) **Extensibilidade**: permitir com que o sistema

tenha suporte para incorporação de novas funcionalidades em função da sua evolução; e (iii) **Reusabilidade**: permitir a reutilização de módulos do sistema em outras aplicações.

4.1. Arquitetura Proposta

A abordagem proposta tem por objetivo apoiar a escolha de desenvolvedores aptos para tarefas de manutenção do tipo adaptativa ou evolutiva, tais como requisições de mudança (*change request*), e do tipo corretiva para problemas expostos pelos relatórios de erros (*bug report*). Este apoio provém da visualização individual de desenvolvedores com suas respectivas sugestões de tarefas e da visualização específica de tarefas com suas respectivas sugestões de desenvolvedores. As sugestões apresentadas são provenientes da verificação de *expertises* atribuídas em comum entre usuário e tarefa. As *expertises* dos usuários são provenientes das contribuições advindas de plataformas externas ou localmente, a reputação do usuário é obtida em um módulo à parte e o esquecimento é tratado na etapa de visualização de perfil. A visão geral da arquitetura está apresentada na Figura 1.

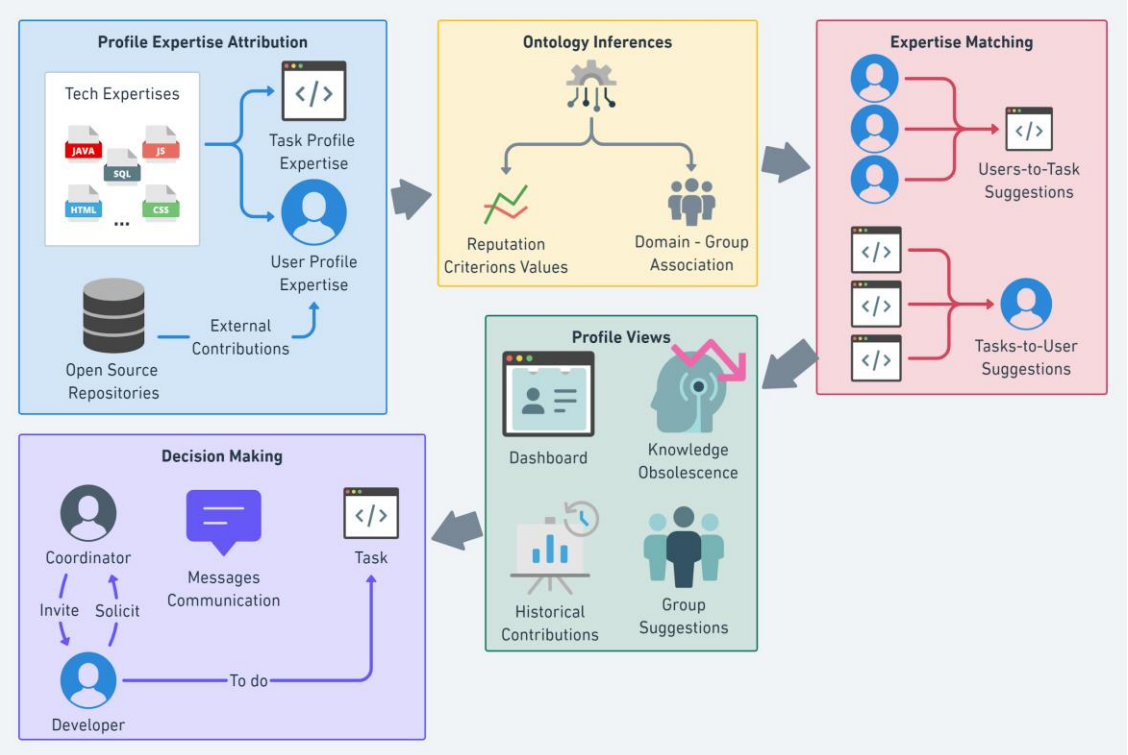

Figura 1. Visão geral da Arquitetura proposta.

O módulo *Profile Expertise Attribution* é responsável pela atribuição de *expertises* de tecnologias para as tarefas registradas na plataforma e o preenchimento do perfil de *expertises* dos usuários. Os desenvolvedores que possuem contribuições em outras plataformas têm a opção de preencherem o perfil com o registro das *expertises* provenientes de suas contribuições externas. Essas contribuições podem ser *commits* em repositórios públicos de código fonte no *GitHub*, tarefas de desafios em desenvolvimento e manutenção de software no *TopCoder*, e respostas dadas a perguntas no *StackOverflow*. Para a constituição do perfil do usuário, os registros das contribuições externas obtidas devem possuir valores de data e *expertise(s)* associadas, para posterior tratamento em relação ao esquecimento.

O módulo *Ontology Inferences* tem como propósito dar apoio à formação de grupos de usuários desenvolvedores que possuem *expertises* com os domínios de conhecimento associados e valores de reputação atribuídos às suas contribuições passadas. O módulo *Expertise Matching* tem por objetivo fornecer para cada nova tarefa uma lista sugestiva de desenvolvedores aptos, e também para cada desenvolvedor uma sugestão de tarefas relevantes. A identificação dos usuários capacitados para tarefas tem princípio na comparação do perfil de *expertise* de cada tarefa com o perfil de cada desenvolvedor. A abordagem faz então a identificação das *expertises* em comum entre cada tarefa para com um usuário, e também entre cada usuário para com uma tarefa.

No módulo *Profile Views*, por meio de uma *Dashboard,* cada usuário pode visualizar essas sugestões de tarefas de acordo com suas *expertises*, assim como cada dono de tarefa pode visualizar as sugestões de usuários com *expertises* relevantes para as tarefas. Outras opções de visualizações incluem o esquecimento (*Knowledge Obsolescence*), a sugestão de formação de grupos (*Group Suggestions*) e o histórico de contribuições (*Historical Contributions*) que são exibidas para cada *expertise* associada. O módulo *Decision Making* representa o momento em que usuários desenvolvedores e coordenadores de tarefas interagem para tomada de decisão. Seja um desenvolvedor que vê uma sugestão de tarefa e decide solicitar ao coordenador desta para contribuir, ou então um coordenador que visualiza o perfil de um desenvolvedor sugerido e decide convidá-lo para contribuir na tarefa.

4.2. Implementação da Arquitetura em um Ambiente Convencional

Conforme a arquitetura apresentada pela Figura 1, o usuário que se cadastra no sistema pode atuar de duas formas diferentes: sendo um desenvolvedor a procura de tarefas condizentes com seus conhecimentos, ou então como um coordenador de tarefas à procura de desenvolvedores aptos nas *expertises* para suas tarefas criadas.

A aplicação fornece as seguintes opções para o usuário: elaborar seu perfil de *expertises*, criar tarefas e nestas associar as *expertises* necessárias, e gerenciar as equipes criadas para resolução de suas tarefas elaboradas e ver outras equipes que o próprio esteja participando como desenvolvedor. Em cada tarefa criada existe a opção de o coordenador procurar por sugestões de desenvolvedores, devidamente registrados, que possuam as mesmas *expertises* da tarefa em seus perfis. O desenvolvedor, após o registro de suas próprias *expertises* em seu perfil, tem a opção de solicitar a busca por tarefas com as mesmas *expertises*.

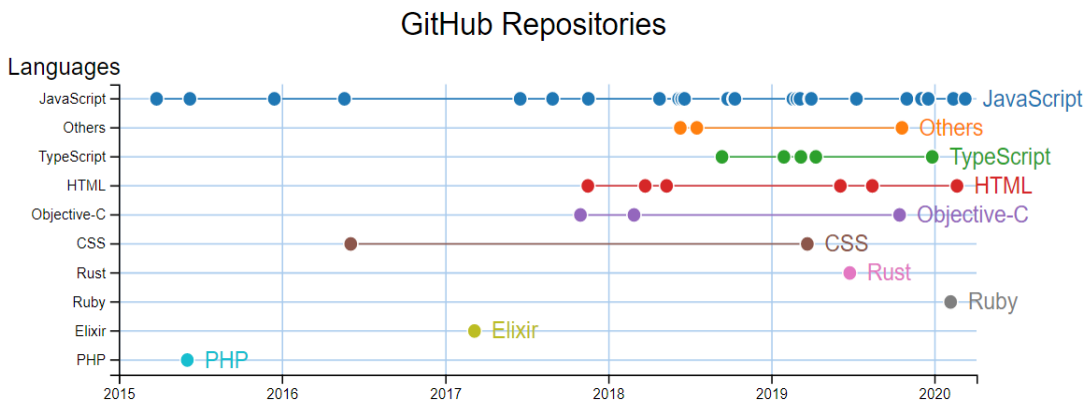

Figura 2. Gráfico exibindo repositórios criados por um usuário do *GitHub*.

Na elaboração do perfil do desenvolvedor, tem-se a opção de registro de acesso às plataformas externas em que o respectivo usuário possua conta. Este, ao disponibilizar seu *username* do *GitHub* (2020) seu *handle* do *TopCoder* (2020) ou seu *Id* do *StackOverflow* (2020) permite ao sistema acessar dados de suas respectivas contas disponíveis publicamente pelas próprias plataformas através de suas APIs. Com os dados disponíveis de contribuições passadas, associados a algum registro de *expertise* (por meio de *tags*, *skills*, ou nome de linguagens usadas nos repositórios), há a opção de exibir, através de um gráfico, os registros das contribuições efetuadas por *expertise* ao longo do tempo. A Figura 2 ilustra esse gráfico no qual cada círculo preenchido representa os repositórios de um usuário do *GitHub* em ordem de data de criação da esquerda para a direita[2].

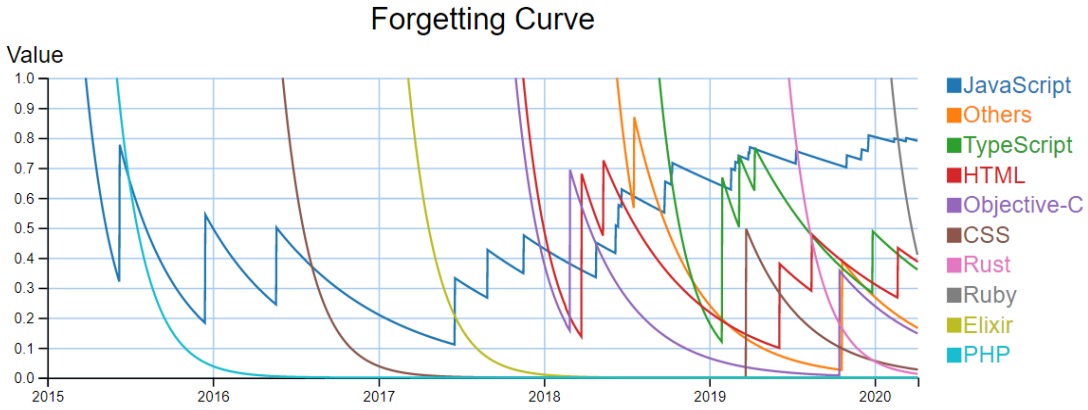

Figura 3. Gráfico exibindo curva de esquecimento das *expertises* de um usuário do *GitHub*.

A Figura 3 exibe um gráfico que mostra o esquecimento associado às diferentes tecnologias associadas aos repositórios ilustrados na Figura 2. No exemplo das Figuras 2 e 3, nota-se que o usuário do *GitHub* possui mais contribuições de repositórios criados na linguagem *(expertise)* JavaScript. A cada nova interação efetuada (novo repositório criado), há um incremento na curva de esquecimento para simular o reforço de memória na *expertise*, e o decaimento posterior passa a ser menor do que antes. Havendo poucas interações e estas sendo antigas, o conhecimento retido tende à zero por causa do esquecimento aplicado. No exemplo do usuário do *GitHub*, este não possui mais relevância para com a linguagem PHP por ter interagido apenas uma vez no passado distante.

Apesar de ser observada na solução proposta a satisfação de alguns requisitos funcionais, e alguns não funcionais, para o suporte à localização de especialistas em bases de dados globalmente distribuídas, alguns pontos relacionados à escalabilidade necessitam ser avaliados. Para tanto, a proposta foi avaliada em um ambiente de alto desempenho para obter evidências em relação ao processamento de bases distribuídas. Isto permitirá ampliar a localização de especialistas com diferentes perfis para apoiar a manutenção de sistemas complexos. A Seção 5 apresenta a avaliação da solução proposta em um Ambiente de Alto Desempenho.

[2] Visualizações associadas a outras bases de dados não foram apresentadas por restrições de espaço.

5. Avaliação da Proposta (Ambiente e Resultados Experimentais)

Nesta seção, são apresentados os cenários de alto desempenho utilizados como suporte para avaliar o uso da arquitetura desenvolvida e os seus resultados experimentais em uma configuração de *clusters* distribuídos.

5.1. Planejamento do Experimento

O experimento[3] foi planejado para avaliar a necessidade de um suporte computacional capaz de apoiar o processamento de diversas bases abertas, considerando a *performance* e a escalabilidade da abordagem. Para simular a obtenção das contribuições externas em diversas bases, dados de contribuições de usuários dessas bases foram extraídos e salvos, e durante o experimento, esses dados foram replicados para simular a leitura e processamento de uma quantidade maior de usuários e suas contribuições em diversas bases. Assim, o experimento busca avaliar o desempenho dessa leitura e processamento dos dados de contribuições para agrupá-los por *expertise*.

5.2. Ambiente Computacional

O ambiente distribuído paralelo utilizado neste trabalho foi uma configuração do Grid5000 (2020), caracterizado por 3 *clusters*, 40 nodos, 1408 cores, atingindo um potencial computacional de 90.7 TFLOPS.

O primeiro *cluster*, denominado de Dahu, com 32 nodos, sendo 2 x Intel Xeon Gold 6130, 16 cores/CPU, com 192 GiB de memória, capacidade de armazenamento de 240 GB SSD (+ 480 GB SSD e + 4.0 TB HDD) e uma rede de 10 Gbps + 100 Gbps Omni-Path.

O *cluster* Troll, com 4 nodos, 2 x Intel Xeon Gold 5218, 16 cores/CPU, com 384 GiB de memória e 1.5 TiB PMEM, com capacidade de armazenamento de 480 GB SSD (+ 1.6 TB SSD) e uma rede 10 Gbps + 100 Gbps Omni-Path.

O terceiro nodo computacional, chamado de Yeti, com 4 nodos, 4 x Intel Xeon Gold 6130, 16 cores/CPU, com 768 GiB de memória, com capacidade de armazenamento de 480 GB SSD (+ 3 x 2.0 TB HDD e + 1.6 TB SSD) e uma rede 10 Gbps + 100 Gbps Omni-Path.

O ambiente convencional utilizado para este experimento foi um computador pessoal com processador Intel Core i7 2860QM, com 8 GB de memória RAM e capacidade de armazenamento de 750 GB em HD SATA (7200 RPM).

5.3. Resultados Experimentais

Nesta seção serão apresentados os resultados considerando-se um ambiente de alto desempenho. Como apresentado em Pioli et al. (2019), as melhorias nos ambientes de E/S têm uma importância que vêm se consolidando de uma maneira diferencial para uma série de aplicações (exemplos são aquelas com granularidade fina).

A proposta de análise considerou as bases anteriormente mencionadas: *GitHub* (2020), *TopCoder* (2020) e *StackOverflow* (2020). Uma única execução do experimento foi realizada para cada *cluster* e os tempos de leitura e processamento dos dados foram

[3] https://github.com/nathan2m/experiment_grid5000

registrados nas tabelas abaixo. As Tabelas 1, 2 e 3 apresentam os resultados referentes à execução da experimentação considerando os nodos dos clusters anteriormente mencionados e a Tabela 4 exibe os resultados da execução considerando o ambiente convencional.

Tabela 1- Resultados da execução no cluster dahu

	github	stackoverflow	topcoder
real	0m33.479s	0m4.954s	0m4.879s
user	0m40.803s	0m5.261s	0m5.671s
sys	0m2.004s	0m0.663s	0m0.845s

Tabela 2 - Resultados da execução no cluster troll

	github	stackoverflow	topcoder
real	0m32.327s	0m4.940s	0m4.886s
user	0m39.555s	0m5.130s	0m5.455s
sys	0m1.902s	0m0.732s	0m1.006s

Tabela 3 - Resultados da execução no cluster yeti

	github	stackoverflow	topcoder
real	0m34.416s	0m5.059s	0m5.011s
user	0m43.792s	0m5.321s	0m5.964s
sys	0m2.403s	0m0.774s	0m0.978s

Tabela 4 - Resultados da execução no ambiente convencional

	github	stackoverflow	topcoder
real	0m57.959s	0m15.902s	0m15.824s
user	0m47.797s	0m9.781s	0m11.031s
sys	0m6.328s	0m3.688s	0m4.359s

De acordo com as Tabelas 1, 2 e 3 podemos perceber que os tempos de execução nos três *clusters* são próximos entre si, e menores do que o tempo de execução no ambiente convencional (Tabela 4). Entretanto, observamos que os tempos obtidos na plataforma do *GitHub* foram superiores nas três opções: *real*, *user* e *sys*. Isso se deve ao fato de que o código construído para consumo de dados da API do *GitHub* explora uma quantidade maior de informações na plataforma. Cabe ressaltar que a opção *real* se refere ao tempo entre a execução e conclusão do comando, a opção *user* se refere ao tempo do usuário no processador e a opção *sys* é o tempo para o *kernel* executar o comando.

Considerando que os dados obtidos tanto da plataforma *TopCoder* quanto do *StackOverflow* são menos explorados, em função da demanda da aplicação construída, observamos que os valores de tempo nas três opções (*real*, *user* e *sys*) estão próximos. Portanto, experimentos adicionais podem ser executados com uma quantidade maior de registros no sentido de analisarmos as variações dos tempos em função da quantidade de registros processados.

5.4. Avaliação dos Resultados

Através dos resultados obtidos é possível afirmar que o suporte de localização de especialistas em bases distribuídas pode ser ampliado para um ambiente computacional de alto desempenho. Com isso, a escalabilidade de poder computacional pode apoiar a utilização de um conjunto diversificado de bases de dados associadas às plataformas de projetos. Adicionalmente, há indícios de que a arquitetura oferece o suporte para atender as buscas por especialistas no contexto de manutenção de sistemas complexos.

Entretanto, obter contribuições das plataformas externas não garante que as informações de *expertises*, reputação e esquecimento dos desenvolvedores sejam condizentes com a realidade observada. Para mitigar este aspecto, a abordagem desenvolvida deve ampliar o suporte à implementação de outras plataformas externas

para busca de informações em contextos diversificados para apoiar a manutenção de software. Experimentos adicionais necessitam ser realizados no sentido de obter evidências sobre as formas pelas quais os dados obtidos em outras plataformas podem enriquecer as indicações de especialistas. Como resultado, o suporte a decisões poderá ser ampliado considerando a complexidade dos sistemas de software contemporâneos.

6. Conclusão

Este trabalho apresentou a proposta de uma abordagem integrada ao contexto de desenvolvimento global de software para a busca de programadores capacitados em *expertises* considerando reputação e esquecimento em diferentes bases de dados abertas.

Ao atentar nas contribuições passadas de um desenvolvedor para elaborar seu perfil de *expertise*, busca-se considerar o fator esquecimento. Isso de modo a minimizar a atribuição de pessoas com conhecimentos obsoletos na realização de manutenção e desenvolvimento de software, e assim buscar aqueles que são experientes e estão ativos. Para tanto, uma solução foi proposta utilizando-se um ambiente computacional convencional e, em seguida um experimento foi realizado em outro ambiente de alto desempenho, este último apresentou um diferencial em relação ao processamento das bases de dados distribuídas. Como resultado, há indícios de que a solução proposta necessita ser explorada neste último ambiente com o objetivo de estender a quantidade de bases utilizadas para apoiar a manutenção de sistemas complexos. Espera-se com isso que uma quantidade maior de especialistas possa ser analisada para uma diversidade de sistemas.

A contribuição deste trabalho consiste de uma solução com capacidade de ser estendida para um ambiente computacional de alto desempenho para apoiar as tarefas de busca por desenvolvedores capacitados em *expertises*. A abordagem desenvolvida apresenta recursos de visualizações para apoiarem a tomada de decisões relacionadas à evolução da curva de esquecimento no contexto de cada *expertise*. Ainda, tem-se o apoio à formação de grupos de desenvolvedores com diferentes especialidades para atuarem em uma mesma tarefa.

Devido à necessidade de uma escalabilidade de poder computacional, nossa abordagem foi executada em um ambiente distribuído de alto desempenho, o Grid5000, que nos apresentou uma indicação de que a arquitetura proposta pode escalar para uma diversidade de bases de dados. Como resultado, as buscas por especialistas podem ser estendidas para diferentes plataformas capazes de fornecerem informações para ampliar o suporte à decisão no contexto da manutenção de software.

Como trabalhos futuros pretende-se evoluir a solução apresentada para outras bases de dados, considerando, sobretudo os aspectos sociais envolvendo o desenvolvimento de sistemas complexos. Neste contexto, cabe explorar as plataformas sociais e relacionar as informações obtidas com as plataformas de projeto de software, com o objetivo de apoiar a manutenção de sistemas complexos.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001. Agradecemos também ao CNPq, INESC Brasil e EDP Projeto SIGOM que apoiam parcialmente este trabalho.

Experimentos apresentados neste trabalho foram realizados utilizando recursos do Grid'5000, que é desenvolvido como parte do projeto INRIA ALADDIN, com suporte do CNRS, RENATER e várias universidades, assim como outras organizações de fomento (ver https://www.grid5000.fr).

Referências

De Neira, A. B., Steinmacher, I., Wiese, I.S. (2018) Characterizing the hyperspecialists in the context of crowdsourcing software development. Journal of the Brazilian Computer Society, v. 24, n. 1, p. 17.

Erlikh, L. (2000) "Leveraging legacy system dollars for e-business". IT professional, v. 2, n. 3, p. 17-23.

GitHub (2020): https://developer.github.com/v3/ , Acesso em 11 de agosto de 2020

Goyal, A., Sardana, N. (2017) "Machine Learning or Information Retrieval Techniques for Bug Triaging: Which is better?" e-Informatica Software Engineering Journal, v. 11, n. 1.

GRID5000 (2020): https://www.grid5000.fr, Acesso em 11 de agosto de 2020

Hattori, L. P., Lanza, M., Robbes, R. (2012) "Refining code ownership with synchronous changes". Empirical Software Engineering, v. 17, 4-5, p. 467-499, 2012.

Khatun, A., Sakib, K. (2016) "A bug assignment technique based on bug fixing *expertise* and source commit recency of developers". In: Computer and Information Technology (ICCIT), 2016 19th International Conference on. IEEE, p. 592-597.

Lélis, C. A. S. et al. (2016) "ArchiRI-uma arquitetura baseada em ontologias para a troca de informações de reputação". In: Anais do XII Simpósio Brasileiro de Sistemas de Informação, p. 060-067.

Miguel, M. A. et al. (2016) "A framework to support effort estimation on software maintenance and evolution activities". In: Proceedings of the XII Brazilian Symposium on Information Systems, p. 31.

Oliveira Jr, M. et al. (2019) "Recommending External Developers to Software Projects based on Historical Analysis of Previous Contributions". In: Proceedings of the XXXIII Brazilian Symposium on Software Engineering. p. 417-426.

Pioli, L., Ströele, V., Dantas, M. A. R., (2019) "Research Characterization on I/O Improvements of Storage Environment" International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 9p. : 287-298

StackOverflow (2020): https://api.stackexchange.com/, Acesso em 11 de agosto de 2020

TopCoder (2020): https://tcapi.docs.apiary.io/, Acesso em 11 de agosto de 2020

Trainer, E.H., Redmiles, D.F. (2018) "Bridging the gap between awareness and trust in globally distributed software teams". Journal of Syst. and Software, 144, p. 328-341.