PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

Licença Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported

Assistance:*

Um middleware de computação oportunística em grades computacionais com escalonamento distribuído de tarefas

José F. R. Fonseca[1]
Luís Fabrício Goés[2]

Resumo

Este artigo apresenta o sistema Assistance, um *middleware* desenvolvido pelos autores para grades computacionais oportunísticas. Grades computacionais são soluções eficientes para otimizar o uso de recursos como memória e processadores, minimizando investimentos em novo *hardware*. Tarefas são escalonadas em recursos ociosos de forma oportunística, reduzindo a carga em um nó da grade. Soluções atuais não oferecem flexibilidade na política de escalonamento e utilização de recursos o suficiente, o que inviabiliza seu uso em organizações. O sistema Assistance foi desenvolvido neste trabalho como uma nova solução para esses e outros desafios da área de grades computacionais. Outros desafios incluem a interoperabilidade entre *middlewares* distintos, a compatibilidade com códigos executáveis otimizados para uma plataforma específica, e a segurança dos dispositivos da rede. Descrevem-se aqui o funcionamento e o protocolo de comunicações do Assistance, bem como a forma como algoritmos customizados de escalonamento oportunístico podem ser implementados no sistema. Um panorama de trabalhos relacionados evidencia o diferencial do Assistance, de não envolver a submissão de códigos executáveis para a realização de tarefas. Experimentos para a avaliação da eficiência do Assistance consistiram na execução distribuída de aplicações desenvolvidas pelos autores e executadas numa grade de 4 servidores com hardware homogêneo e diferentes combinações de políticas de escalonamento. O tempo de resposta das aplicações quando executadas de forma serial foi até 82,76% maior do que quando executadas de forma distribuída pelo Assistance.

Palavras-chave: grade computacional oportunística, escalonamento distribuído, grade computacional, Assistance, *middleware*.

*Artigo apresentado à Revista Abakos

[1]Graduando, Programa de Graduação em Ciência da Computação da PUC Minas, Brasil.
[2]Professor Orientador, Programa de Graduação em Ciência da Computação da PUC Minas, Brasil.

Abstract

This paper presents Assistance, an opportunistic grid computing middleware developed by the authors. Opportunistic grids are efficient solutions to optimize the usage of computing resources, like memory and processors, minimizing the need for new hardware. Computing tasks are opportunistically scheduled on idle resources, reducing the total load in each point of the grid. Current solutions do not support enough flexibility on brokering policies to adapt to resource usage rules of organizations. Assistance was developed as a solution for those and other challenges of grid computing, like interoperability of middlewares, compatibility of executable code optimized for a specific platform, and cybersecurity. We describe the architecture, communication protocols and functioning of Assistance, as well as the steps to install customized opportunistic brokering policies. An analysis of related work evidences the differential of Assistance, of avoiding the transmission of executable code for task completion. Experiments evaluate the efficiency of Assistance on a grid of four homogeneous servers using different brokering policies. The serial execution time of test applications was up to 82.76% bigger than the grid response time for the same applications.

Palavras-chave: opportunistic grid computing, distributed brokering, grid computing, Assistance, middleware.

1 INTRODUÇÃO

Aplicações muitas vezes deixam de utilizar por completo os recursos computacionais, como memória e processadores, dos dispositivos em que são executadas. Em outras situações, os recursos disponíveis podem ser insuficientes para garantir o nível de qualidade de serviço adequado para o usuário das aplicações. Grades computacionais oportunísticas são soluções de abordagem distribuída que permitem o compartilhamento de recursos entre diferentes dispositivos através de uma rede, facilitando modularidade e escalabilidade de sistemas e reduzindo os custos de implantação e manutenção (FOSTER; KESSELMAN; TUECKE, 2001; MCCOOL; REINDERS; ROBISON, 2012). Recursos ociosos podem ser usados para executar aplicações de parceiros remotos, sem prejudicar a qualidade do serviço prestado ao proprietário dos recursos.

Uma grade computacional é definida pelos dispositivos que fazem parte da mesma, interligados por um sistema distribuído presente em todos os dispositivos, denominado *middleware*. Tais dispositivos podem pertencer a vários indivíduos ou organizações, denominados *entidades*. A disponibilidade e autenticidade do uso dos recursos e a segurança das informações da entidade que os disponibiliza em uma grade são problemas que devem ser abordados pelo *middleware* (KSHEMKALYANI; SINGHAL, 2008; CEDERSTRÖM, 2010). Um *middleware* intermedeia o processo em que um dispositivo submete uma tarefa, uma tupla que relaciona uma aplicação a um conjunto de dados que a mesma deve utilizar, para outro dispositivo da grade. O *middleware* de uma grade deve oferecer uma solução para a execução de *tarefas* em diferentes dispositivos, que podem usar diversas arquiteturas de processamento e memória, abordando dificuldades de implementação de aplicações multiplataforma. A alocação de recursos compatíveis para a execução de uma tarefa, e a ordenação das tarefas a serem executadas, processo denominado *escalonamento*, é um problema não-trivial que afeta o desempenho de uma grade.

Para o suporte da computação oportunística em grades computacionais, foi desenvolvido neste trabalho o sistema Assistance, um *middleware* para grades computacionais oportunísticas que permite o compartilhamento seguro e eficiente dos recursos de diferentes indivíduos e organizações. O Assistance permite customização das políticas de escalonamento em cada nó da grade, para atender aos critérios de uso de recursos de diferentes entidades. Diferente de outros *middlewares*, o Assistance não requer que um dispositivo da grade envie códigos executáveis para outro para a realização de uma tarefa. Tal característica contribui para a segurança do sistema, por dificultar a disseminação de códigos maliciosos na grade. A decisão de evitar a submissão de códigos executáveis de um dispositivo da grade para outro também tem por objetivo suportar o uso de códigos otimizados para uma plataforma na execução de uma tarefa.

Sendo A e B dois dispositivos de uma grade Assistance, o sistema permite que A submeta para B um conjunto de dados que B deve utilizar na computação de um determinado algoritmo, e B deve retornar para A os resultados. Um dispositivo B somente pode se comprometer a atender a requisição de A caso B possua uma implementação para o algoritmo requerido por A. Muitos dispositivos possuem implementações para os mesmos algoritmos, como por

exemplo as ferramentas de ordenação da biblioteca java.utils (GOSLING; MCGILTON, 2003). Portanto, o Assistance permite que um usuário utilize apenas recursos e códigos previamente disponibilizados em dispositivos da grade para realizar suas tarefas. Tal característica permite independência entre uma tarefa e os códigos necessários para realizar a mesma, possibilitando o uso de códigos otimizados e compatíveis com o dispositivo que os executa, o que é ideal em um ambiente de dispositivos heterogêneos.

Este trabalho teve por objetivo desenvolver e apresentar uma nova abordagem ao problema do compartilhamento de recursos computacionais. É do escopo deste trabalho atestar a funcionalidade da solução apresentada e apresentar evidências sobre o desempenho da solução em diferentes configurações para diferentes tipos de aplicações. Tais evidências devem orientar desenvolvedores de grades computacionais oportunísticas com políticas de escalonamento customizadas, e pesquisadores estudando o comportamento de grades com diversas políticas de escalonamento. Não é do escopo deste trabalho melhorar a eficiência de algoritmos existentes, nem propor novo algoritmo de escalonamento, nem reproduzir ou melhorar resultados de outros trabalhos.

Este artigo está organizado em 7 seções. A seção 2 apresenta os conceitos de grades computacionais oportunísticas, escalonamento oportunístico distribuído e trabalhos relacionados. A seção 3 descreve o funcionamento do sistema Assistance, sua arquitetura e protocolo de comunicações, e detalhes sobre sua implementação. A seção 4 descreve os testes realizados para avaliar a plataforma, sendo os resultados apresentados na seção 5, e as conclusões e trabalhos futuros na seção 6.

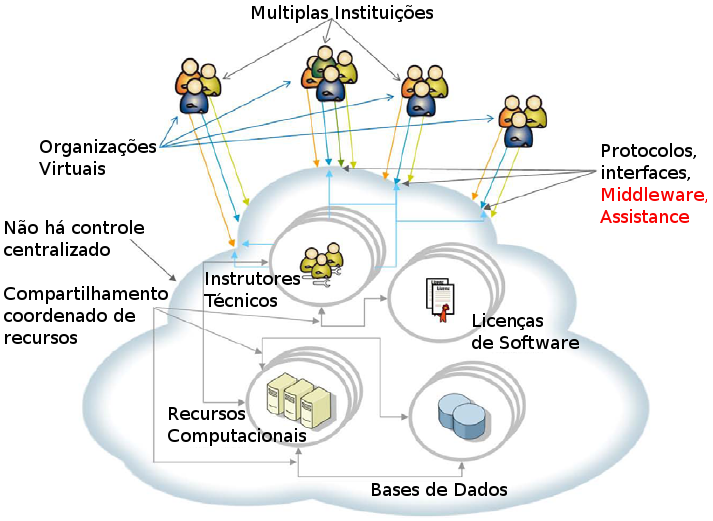
2 REVISÃO BIBLIOGRÁFICA

2.1 Computação em Grade

O termo *computação em grade* normalmente se refere a sistemas distribuídos que permitem controle da qualidade de serviço na solução de problemas, por meio do compartilhamento coordenado, flexível e seguro de recursos em organizações virtuais, que são coleções dinâmicas de indivíduos e instituições independentes, proprietárias e gestoras de processadores, memória, sensores e dados (FOSTER; KESSELMAN, 2010; FOSTER; KESSELMAN, 2004; VAQUERO et al., 2008; KAHANWAL; SINGH et al., 2013). A interface de compartilhamento deve seguir padrões e protocolos públicos, conhecidos por todos os participantes. Tal sistema de comunicação é denominado *middleware*, e não deve ser baseado na coordenação centralizada dos recursos (FOSTER; KESSELMAN; TUECKE, 2002; FOSTER et al., 2008).

A Figura 1, ilustração de Schwiegelshohn et al. (2010) para o modelo de Foster, Kesselman e Tuecke (2001), traduzida, representa o modelo de grade. Um recurso frequentemente utilizado em grades computacionais é a transferência direta de dados entre máquinas, evitando centralização das comunicações em um sistema intermediário, o que potencializa a escalabili-

Figura 1 – Arquétipo do modelo de uma grade computacional



Fonte: Adaptado de (SCHWIEGELSHOHN et al., 2010)

dade. Esse é o modelo ponto-a-ponto (*peer-to-peer* - P2P) (TANENBAUM, 2011; FOSTER; IAMNITCHI, 2003).

O usuário de uma grade pode ter o papel de cliente, caso no qual submete à grade requisições (*tasks*, do inglês, tarefas), ou de servidor, que as executa em seus recursos. Uma tarefa pode ser definida como uma tripla contendo: (1) códigos de um programa, (2) um conjunto de dados sobre os quais o programa deve ser executado, e (3) um conjunto de especificações que descrevem a plataforma e método de execução do programa. A arquitetura de grades computacionais pode incluir um sistema de *brokering*, ou *escalonamento*, que determina quais servidores de uma grade estão aptos a realizar cada tarefa submetida, e como estas devem ser ordenadas. A presença de um escalonador é marca de uma arquitetura que contém um serviço que intermedeia o trânsito de tarefas entre clientes e servidores, gerenciando sua execução e eventuais falhas. Existem diversos algoritmos para otimizar a atividade de escalonamento, e os que foram utilizados neste trabalho estão explicados na próxima seção (XHAFA; ABRAHAM, 2010).

Uma grade computacional pode ser usada para minimizar custos operacionais de uma instituição, reduzindo investimentos em estabelecimento, manutenção e expansão, assegurando a disponibilidade de sistemas e facilitando o acesso aos mesmos. A interação entre diferentes *middlewares*, tarefa complexa nas grades atuais, potencializa todos esses benefícios (SCHWIEGELSHOHN et al., 2010; SNAVELY et al., 2003). Preocupações únicas emergem no contexto de grades, como compatibilidade dos códigos de uma tarefa com as plataformas em que pode vir a ser executada. Tal problema é frequentemente solucionado com o uso de máquinas virtuais, abordagem que simultaneamente facilita a compatibilidade de códigos e aumenta o controle do proprietário de um recurso computacional sobre o uso do mesmo, ao custo de flexibilidade e otimização de sistema para sua plataforma. A gestão de máquinas virtuais é um desafio das grades computacionais, juntamente com o controle preciso da quantidade de recursos alocados para cada cliente, a gestão da energia elétrica e do fluxo das redes de comunicações.

A exploração dos recursos computacionais domésticos não é um problema trivial, mas pode render significativos resultados (ANDERSON, 2003). A esse tipo de grades dá-se o nome

de *Desktop Computer Grids* (DCG) (ZHAO; LIU; LI, 2011). Diversas soluções propostas para DCG estão descritas na seção "trabalhos relacionados". Algumas dessas soluções utilizam o conceito de *computação oportunística*.

O princípio da computação oportunística é a alocação de tarefas em recursos que não estejam sendo utilizados. O uso de recursos ociosos apresenta alguns obstáculos específicos. Por exemplo, o proprietário dos recursos pode os requerer enquanto estiverem ocupados com a tarefa de um terceiro, ou tais recursos podem se tornar indisponíveis por alguma razão técnica, como uma falha da rede ou falta de energia. A localização dos recursos apropriados para uma tarefa em si é um desafio, pois exige conhecimento sobre a compatibilidade dos códigos de uma tarefa com os recursos de vários dispositivos. Contudo, a otimização de uma grade por meio do uso de recursos ociosos é uma forma eficiente de aumentar a taxa de processamento da grade, sem que investimentos em hardware sejam necessários.

2.2 Escalonamento de Tarefas

Diversas políticas e algoritmos lidam com os obstáculos do escalonamento em grades, como os apresentados por Subramani et al. (2002). Políticas de escalonamento centralizadas, em que um dispositivo gerencia a alocação de recursos em toda a grade, têm desempenho superior a políticas distribuídas, contudo, são menos escaláveis. Políticas distribuídas estão sujeitas a problemas de súbita indisponibilidade dos recursos utilizados, problemas que podem ser contornados pelo uso de replicação das tarefas requisitadas. A replicação de tarefas consiste na alocação das mesmas em diversos grupos de recursos, simultaneamente, para o caso de que algum falhar.

Políticas de escalonamento oportunístico distribuído podem buscar alocar tarefas nos recursos menos utilizados de toda a grade, ou nos que sejam utilizados apenas em horários específicos. Tarefas advindas de certas origens, inclusive da mesma rede local que os recursos, podem ser escalonadas prioritariamente. Um exemplo de um algoritmo complexo de escalonamento distribuído é o EDFSRT, proposto por Sharma e Mittal (2013), em que tarefas são escalonadas dependendo da proximidade com o horário até o qual as mesmas são relevantes (*deadline*), e do tempo restante para a sua conclusão.

Tanenbaum (2009) cita políticas de escalonamento que podem ser utilizadas no contexto de escalonamento oportunístico distribuído, como por exemplo *First-in-First-Out* (FIFO) e *Shortest Job First* (SJF).

No escalonamento pelo algoritmo FIFO, é determinado um número máximo de tarefas que podem ser executadas concorrentemente, e quaisquer tarefas recebidas após o alcance de tal número serão acumuladas numa fila para serem executadas, em ordem de chegada, a medida que aquelas em execução vão sendo concluídas. Esse algoritmo busca minimizar o tempo de execução de uma tarefa, dedicando recursos exclusivamente, ao custo de aumentar o tempo de espera em filas.

No escalonamento SJF, um número máximo de tarefas que podem ser executadas concorrentemente será fixado, e quaisquer tarefas recebidas após o alcance de tal número serão acumuladas numa fila. Ao concluir uma das tarefas em execução, será selecionada a tarefa na fila que tem o menor tempo de execução, tomando como base a natureza da mesma ou o histórico de desempenho de tarefas semelhantes. Esse algoritmo, idealmente, minimiza o tempo de execução e espera de uma tarefa, mas depende da acurácia da previsão do seu tempo de execução, nem sempre garantida.

2.3 Trabalhos Relacionados

Um dos primeiros sistemas de computação em grade foi o Condor, proposto originalmente por Litzkow, Livny e Mutka (1988). O Condor identifica um sistema com recursos não-utilizados, e aloca tarefas para serem executadas no mesmo em *background*. O sistema Condor foi revisto mais recentemente por Thain, Tannenbaum e Livny (2005). Na estrutura do Condor, um usuário contacta um *problem solver*, um intermediário que opera o sistema para obter a resposta buscada pelo usuário. Tal módulo recorre a um agente, um subsistema de escalonamento que, por meio de um subsistema de *match making*, encontra os recursos adequados para atender a necessidade do usuário. Ao identificar os recursos necessários, o Condor cria uma *execução-sombra* da tarefa executada, que permite sincronismo por meio de pontos de recuperação (*checkpoints*) para as várias instâncias que estão executando a tarefa, utilizando ambientes virtualizados. A arquitetura do Condor foi usada como inspiração para as de trabalhos subsequentes, inclusive a descrita no item Arquitetura da seção Assistance do presente artigo.

O sistema Our Grid, desenvolvido por Cirne et al. (2006), utiliza uma arquitetura parecida com a do Condor, e permite que seus usuários submetam à grade programas em Java ou uma linguagem de domínio desenvolvida para esse propósito, compatível com a máquina virtual JVM. Essa é a máquina virtual mais popular do mundo, presente em quase todos os computadores domésticos (GOSLING; MCGILTON, 2003), e dificilmente tem de ser instalada com o middleware. O Our Grid busca ser o menos intrusivo possível, permitindo que recursos compartilhados na grade sejam utilizados de forma oportunística, sendo ocupados apenas quando seu proprietário não os estiver usando.

Diversas políticas de escalonamento podem ser usadas por uma grade Our Grid. Uma dessas é a política *Workqueue with Replication* (WQR), que não requer conhecimento sobre a disponibilidade de recursos no servidor no qual aloca uma tarefa. WQR distribui em ordem uma lista de tarefas para servidores da grade, selecionando tais servidores randomicamente. Ao atingir o fim da lista, WQR submete novamente as mesmas tarefas para outros servidores, ação denominada *replicação de tarefas*. WQR apresenta desempenho similar a políticas de escalonamento que requerem mais informações sobre os servidores utilizados, evidência da eficiência dos algoritmos de escalonamento que não requerem que servidores divulguem informações

como a quantidade de seus recursos e a ocupação dos mesmos (SILVA; CIRNE; BRASILEIRO, 2003).

O Our Grid não permite que usuários criem suas próprias comunidades restritas, tendo por objetivo permitir colaborações de grandes comunidades de usuários especializados e laboratórios. Essa funcionalidade está presente no sistema XtremWeb (FEDAK et al., 2001), que, no entanto, envolve consideráveis complicações nos quesitos de implementação e implantação.

Cederström (2010) realizou uma revisão sistemática de literatura (*systematic literature review* - SLR) comparando diferentes sistemas de DCG, e uma pesquisa comprovando o interesse da comunidade de empresas desenvolvedoras de software em soluções de DCG, desenvolvendo uma aplicação de teste que utiliza o sistema BOINC (ANDERSON, 2004). Os usuários do BOINC podem escolher entre diversos padrões para submeter e receber tarefas, executadas em uma máquina virtual. Já o projeto Globus (FOSTER; KESSELMAN, 1997) busca integrar outros tipos de grades e serviços em uma DCG, como o compartilhamento de dados.

Os trabalhos correlatos analisados representam o panorama dos sistemas de grades de dispositivos domésticos, e revelam hesitação na adoção de soluções de grade por parte da indústria. A pesquisa realizada por Cederström (2010) cita motivos que coincidem com desafios identificados por Schwiegelshohn et al. (2010) e Zhang, Cheng e Boutaba (2010) - a segurança das informações e dispositivos cedidos por uma instituição, e a compatibilidade, inter-operatividade e complexidade de instalação dos *middlewares*.

O sistema Assistance, desenvolvido neste trabalho, se organiza de forma inspirada na arquitetura do sistema Condor. O Assistance, tal qual o Our Grid, utiliza códigos já presentes nos dispositivos da grade, mas imita a abordagem de BOINC, permitindo ao proprietário do dispositivo determinar as aplicações que serão executadas em seus recursos. Diferentemente dos trabalhos correlatos, Assistance prioriza a flexibilização da política de escalonamento, para que possa ser customizadas pelas entidades que disponibilizam seus recursos na grade.

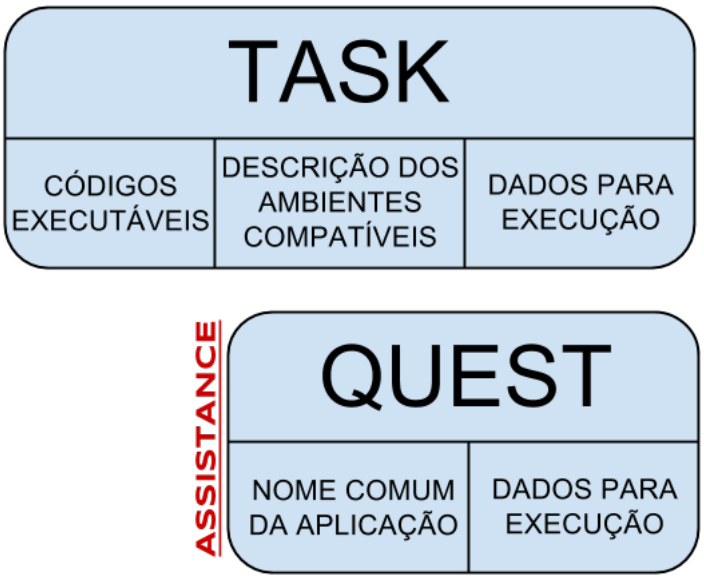
3 ASSISTANCE

3.1 Premissas e Princípios

O Assistance é um sistema de *middleware* para grades computacionais oportunísticas que busca disponibilizar para seus usuários os recursos computacionais presentes em sua rede social. O diferencial do Assistance é que, no lugar de submeter tarefas contendo códigos, descrição de ambientes compatíveis e dados, os usuários submetem "*Quests*", estruturas que contêm somente o *identificador* de uma aplicação comumente conhecida, e os dados que a mesma deve computar. Tais estruturas estão representadas na figura 2 O *identificador* de uma aplicação é um valor-chave único, que deve ser constante entre todos os dispositivos de uma grade Assistance (tal qual um endereço IP).

O Assistance considera que todo usuário tem parceiros que estão conectados e disponí-

Figura 2 – Modelos das estruturas de dados tarefa e *Quest*



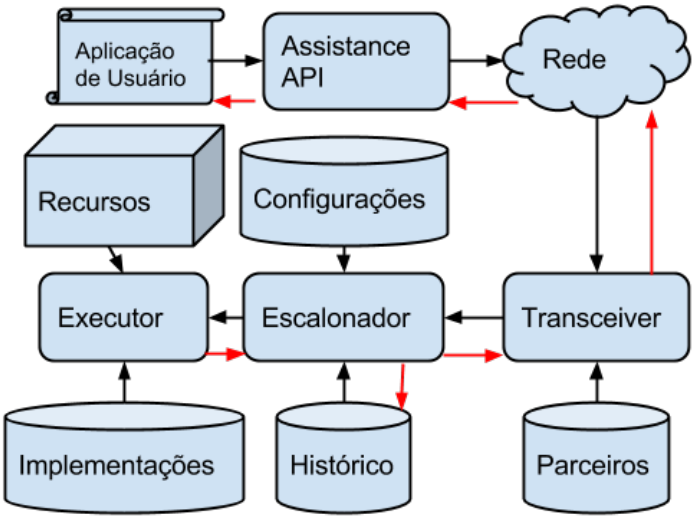
Fonte: próprio autor

veis na rede, cada um utilizando a implementação mais adequada (a seu próprio critério) para as mesmas aplicações. Dispositivos diferentes podem possuir diferentes implementações para as mesmas aplicações, otimizadas para a arquitetura e quantidade de memória, modelo e quantidade de processadores e co-processadores, sensores, e demais recursos do dispositivo. Cada dispositivo na grade tem uma quantidade de recursos suficiente para satisfazer seu proprietário. Contudo, o uso típico de um dispositivo consiste em períodos de baixa necessidade de recursos, entremeados por instantes de mais alta demanda.

Muitas aplicações envolvem a solução dos mesmos problemas e o mesmos algoritmos, que podem estar implementados em diversos dispositivos. Por exemplo, um aplicativo hipotético para *smartphones* que possui o recurso de reconhecimento facial. Um *smartphone* em geral não tem os mesmos recursos que um dispositivo dedicado, como o Microsoft Kinect, desenvolvido especificamente para aplicações que reconhecem traços e partes do corpo (WIKIPEDIA, 2015). Contudo, ambos os dispositivos solucionam o problema do reconhecimento facial, usando o mesmo tipo de dados, no caso, uma imagem.

Em uma situação hipotética, um *smartphone* pede a um Kinect membro da mesma grade Assistance que execute um certo algoritmo de reconhecimento facial em uma imagem. Simultaneamente, o *smartphone* também inicia sua execução do mesmo algoritmo. Neste exemplo, a execução do algoritmo é muito mais rápida no Kinect do que no *smartphone*, devido a seus recursos otimizados para tal aplicação. O Kinect então retorna o resultado das computações para o *smartphone*, que cancela sua execução local ainda não concluída. Assim, o tempo de resposta da aplicação é reduzido pelo uso do Assistance. A execução do algoritmo no *smartphone* foi iniciada concorrentemente com a do Kinect para evitar casos em que este último estivesse ocupado e não pudesse realizar as computações, ou caso a rede estivesse indisponível. Nesses piores casos, o tempo de resposta da aplicação seria o mesmo da execução local dos algoritmos de reconhecimento facial no *smartphone*.

Figura 3 – Diagrama ilustrando a Arquitetura do sistema Assistance



Fonte: Elaborado pelo autor.

3.2 Arquitetura

A arquitetura do Assistance, ilustrada na Figura 3, consiste nos seguintes itens:

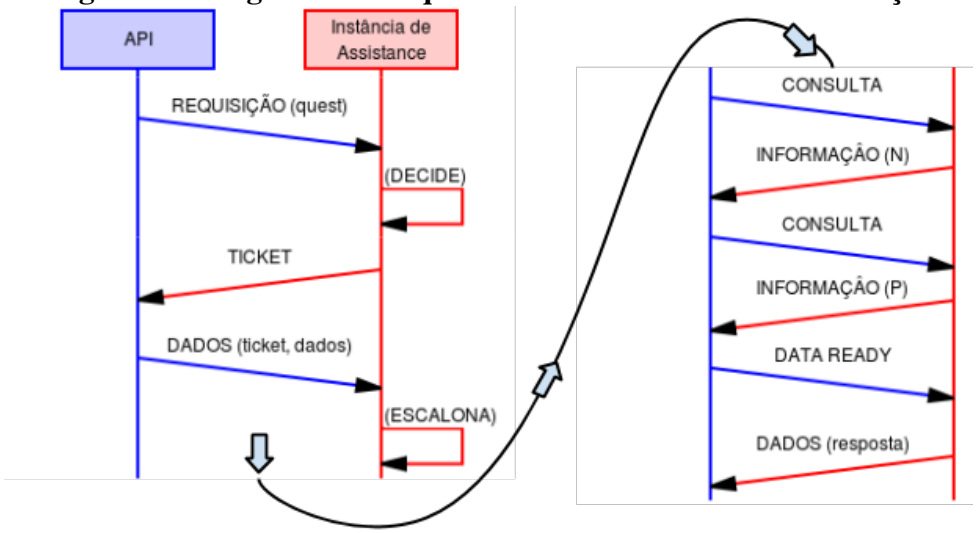
a) Uma interface de aplicação (*application interface* - API), responsável por operar o sistema Assistance, recebe dados de uma aplicação local do usuário, forma uma *quest*, e a submete a todos os parceiros conhecidos. Simultaneamente, a API inicia a execução local da *quest* que foi submetida à grade, para garantir que, no pior caso, o tempo de execução da mesma no sistema Assistance será o tempo de execução serial das *quests* submetidas.

b) O Módulo *Transceiver*, que continuamente monitora a chegada de novas mensagens, é responsável pela transmissão, privacidade e autenticação das mensagens de e para parceiros, e para evitar ameaças à disponibilidade do sistema advindas da rede, como ataques *Denial of Service* (DoS), o *Transceiver* faz uso de um banco de dados de parceiros conhecidos.

c) O módulo Escalonador é responsável por decidir se uma *quest* recebida será executada ou não, e, em caso positivo, a ordenação, prioridade, e demais decisões sobre o escalonamento da mesma. Ele considera uma base de dados de configurações e outra com o histórico de desempenho local e dos parceiros, e registra atualizações no histórico.

d) O módulo Executor faz uso das implementações disponíveis localmente para executar a *quest* nos recursos e é responsável por gerenciar o desempenho da mesma. Ele monitora continuamente os recursos, para que, no caso do crescimento da demanda do usuário local, a execução de uma *quest* externa possa ser interrompida.

Figura 4 – Diagrama de Sequência do Protocolo de Comunicação



Fonte: Elaborado pelo autor.

3.3 Protocolo de Comunicação

As mensagens trafegadas pelo *Transceiver* são encabeçadas pela hora do envio, um valor único que identifica o remetente (seu *token*), e o tipo da mensagem, que pode ser requisição, ticket, consulta, informação, *data ready*, e dados. O protocolo de troca de mensagens está ilustrado, de forma simplificada, na Figura 4.

O protocolo de troca de mensagens do Assistance se inicia no envio de uma mensagem do tipo requisição, que deve conter o nome da aplicação que se deseja a execução, e o tamanho (como quantidade de bytes) dos dados que a mesma deve executar. Tal mensagem deve ser enviada a todos os parceiros conhecidos. Se um dos parceiros estiver em condições de atender a requisição, aproveitará a oportunidade para enviar uma mensagem de ticket. O ticket é um valor encabeçado com o horário de sua criação, combinado com o *hash* SHA-256 da *quest* que o mesmo informa a aceitação. Tal *hash* pode ser assinado pelo parceiro. Ao receber o ticket de uma *quest*, o autor da mesma submete uma mensagem de dados ao autor do ticket, contendo o ticket e os dados que a *quest* requer. Caso o autor da *quest* não receba nenhum ticket em um determinado tempo, poderá tentar novamente submeter a mesma *quest* à grade. Caso o autor do ticket não receba dados até determinado tempo depois de tê-lo emitido, o ticket, e a *quest* a ele associada, serão cancelados. Esse processo permite que um cliente, ao receber mais de um ticket para a mesma *quest*, decida para quais parceiros (dentre os que lhe enviaram um ticket) irá submeter seus dados, ou desista da requisição.

Durante o processamento de uma *quest*, o protocolo de troca de mensagens do Assistance permite que se envie mensagens do tipo consulta, contendo o ticket da *quest* processada, para verificar seu estado. Tais mensagens são respondidas por outras do tipo informação. Se uma mensagem do tipo informação informar ao destinatário que a resposta de sua *quest* foi computada, ele poderá enviar uma mensagem do tipo *data ready* ao remetente, informando que está pronto para receber os dados da resposta. Ao receber tal mensagem, o remetente transmite os dados de resposta de uma *quest*, e finaliza a mesma.

3.4 Características e Implicações

Um modelo de taxonomia de sistemas de grades computacionais, desenvolvido por Zhao, Liu e Li (2011), foi usado para classificar o Assistance pela forma como lida com diversos assuntos. Pode-se considerar os usuários do Assistance como organizados em grupos funcionais, sub-redes não-estruturadas, em que cada usuário pode ter sua própria coleção de contatos. As *tarefas* no Assistance são distribuídas de forma colaborativa, com cada dispositivo identificando localmente os recursos compatíveis disponíveis, e escalonando tarefas para si mesmo. A segurança no Assistance é gerenciada a nível de usuário por meio da autenticação das mensagens trafegadas, e a nível de tarefa por meio da encriptação dos dados e metadados, que advém de uma única fonte. Uma vez que as mensagens são autenticadas, o incentivo à participação na rede tem lugar via reciprocidade. A confiança nos resultados pode ser obtida por verificação dos mesmos por terceiros, ou por replicação do processamento.

As personalizações do Assistance podem ser aplicadas nas configurações e algoritmos internos do escalonador, e na seleção dos parceiros e das implementações disponíveis. O sistema é alheio ao conteúdo das implementações, que, portanto, podem ser escritas em qualquer linguagem de programação, compiladas da forma como for mais conveniente, e fazer uso de quaisquer recursos do sistema. Tal política de uso dos recursos, e por consequência, as regras de escalonamento, pode envolver a ordenação das tarefas para execução, por sua natureza, prioridade informada, horário de criação, horário de chegada, origem, necessidade de recursos e natureza dos mesmos, uso de sensores ou outros dispositivos específicos, uso de coprocessadores, e quaisquer outras informações pertinentes para a política de uso dos recursos da instituição que os mantém. A qualidade do serviço oferecido pode variar com a política de escalonamento utilizada.

A abordagem do uso de *quests* no lugar de tarefas desestimula ações não-autorizadas no sistema, por evitar o tráfego de códigos nunca verificados pelo administrador dos recursos que os executam. Por não estar diretamente relacionado a uma plataforma, recursos ou certa implementação de uma aplicação, o Assistance é idealmente portátil. O sistema é escalável, modular e elástico, não dependendo de um dispositivo central gestor da rede de comunicações, nem um escalonador central gerenciando a execução de processos, permitindo a entrada e saída de nós da grade em qualquer momento.

O Assistance foi elaborado de forma a maximizar a autonomia dos servidores da grade, permitindo que cada um realize seu próprio escalonamento de tarefas. Assim, em uma mesma grade, existe a possibilidade de que diversas políticas de escalonamento estejam sendo utilizadas simultaneamente. Em uma grade Assistance, pode-se estudar os efeitos da interação de diversas políticas de escalonamento, observando o fluxo de processamento de tarefas, os tempos de resposta e execução de cada uma, e a ocupação dos servidores.

3.5 Implementação

O Assistance foi implementado com a linguagem de programação Python, de propósito geral, orientada a objetos, interpretada como *script*, e portátil, pois foi implementada para diversas plataformas (ROSSUM; DRAKE, 2002). Python normalmente não é recomendada para implementar sistemas de alto desempenho, devido a sobrecarga de interpretação; contudo, como demonstraram Cai, Langtangen e Moe (2005), tal dificuldade pode ser contornada, para ganhos de desempenho, com a inclusão de códigos em outras linguagens em partes do projeto. Essa abordagem é utilizada em Assistance, tendo sido o *middleware* do sistema, a aplicação de usuário e a API implementadas na linguagem Python.

Python é recomendada para rápida prototipação de projetos de software, por ser legível mesmo para pessoas sem experiência no uso da mesma, pela sua grande quantidade de bibliotecas de diversas utilidades, e por possuir uma comunidade ativa de usuários para resolver dúvidas e prover exemplos. Assim, é uma linguagem adequada ao rápido desenvolvimento de uma aplicação portátil de múltiplas funções distintas, como o Assistance.

O módulo Executor consiste em uma classe que executa uma *quest* utilizando o terminal do sistema operacional para chamar um *bash script* denominado *AssistanceApp*, tendo como argumentos os dados da *quest*. Cada aplicação executada no Assistance deve possuir seu próprio *script AssistanceApp*, identificado tal qual o nome comum da aplicação requisitada. O *script AssistanceApp* de uma aplicação instrui o sistema operacional a executar a implementação local da mesma sobre os dados passados como parâmetros. Por padrão, o sistema operacional executa cada *AssistanceApp* como um processo concorrente, realizando o escalonamento próprio do sistema operacional entre seus diversos processos. A execução de uma *AssistanceApp* produz um arquivo *stdout* com a saída da execução, e um arquivo *stderr*, com metadados sobre a mesma, ou informações sobre erros de execução. O Executor escreve o arquivo de LOG do sistema, contendo infomações sobre o desempenho de cada *quest* executada.

O módulo Escalonador foi implementado (integralmente em Python) para aceitar quaisquer *quests* recebidas e acumulá-las em um *buffer*. O escalonamento de tarefas consiste em utilizar os algoritmos de escalonamento implementados (FIFO e SJF) para gerenciar a ordem e o momento em que o *script* de uma *AssistanceApp* será chamado. No escalonamentos FIFO e SJF, no máximo 4 tarefas podem ser executadas concorrentemente. Também foi implementado um terceiro escalonamento, no qual todas as *quests* são executadas concorrentemente a partir do momento em que forem recebidas.

O módulo *Transceiver* foi implementado (integralmente em Python) como uma aplicação-servidor continuamente ouvindo as portas TCP 21902 e 21982, respectivamente, para atender a submissão de novas *quests* e transmissão dos resultados. Para este trabalho, o *Transceiver* consegue se comunicar apenas com dispositivos em sua mesma rede local (LAN), não considerando autenticação ou demais fatores de segurança, e aceitando quaisquer mensagens compatíveis com o protocolo do Assistance.

Figura 5 – Legenda para as Configurações da Grade

Fonte: Elaborado pelo autor.

4 METODOLOGIA

4.1 Configuração dos Experimentos

Para testar o desempenho da implementação do sistema Assistance, foram utilizados cinco dispositivos, todos executando o sistema operacional Ubuntu Linux 14.04 e ligados na mesma LAN Ethernet, conectados a um *switch* de 100 Mbps. Cada dispositivo possui um disco rígido SATA-III de 7200RPM, 4094MB de memória primária DDR3-1600MHz e um processador Intel Core i5-4570 de 4 núcleos físicos, operando à 3.20Ghz.

Em quatro dos dispositivos, doravante denominados servidores, foi copiada a mesma versão da implementação do Assistance e das aplicações testadas. No quinto dispositivo, doravante denominado cliente, foram copiadas uma aplicação de testes, os dados necessários para a execução da mesma, a API operadora do sistema, e uma lista com os endereços IP dos servidores.

Foram geradas todas as combinações possíveis de políticas de escalonamento para a grade Assistance, cada qual uma possível configuração da grade, identificada por um número de três dígitos. O primeiro dígito indica quantos servidores escalonam suas tarefas sem limites para o número de processos concorrentes, o segundo quantos utilizam o algoritmo FIFO, limitando a concorrência em 4 *quests*, e o terceiro quantos utilizam o algoritmo SJF, também limitando a concorrência. As políticas de escalonamento são atribuídas para cada servidor seguindo a ordem de seus endereços IP, assim, o servidor com o menor IP será o primeiro a estabelecer sua política, e o de maior IP será o último.

Por exemplo, na configuração 121, o servidor de menor IP escalona suas tarefas sem limitar a concorrência (S.L.C.), dois servidores utilizam FIFO, e o último servidor, de maior IP, utiliza o algoritmo SJF. Tal situação está descrita na figura 5. A soma dos dígitos que identificam uma configuração da grade resultará no número de servidores que fazem parte da mesma. Assim, as 15 combinações possíveis para uma grade de quatro servidores e três políticas de escalonamento são: 004, 013, 022, 031, 040, 103, 112, 121, 130, 202, 211, 220, 301, 310 e 400.

Em cada teste realizado, em cada configuração, um único cliente submeteu todas as *quests* de cada aplicação para cada servidor, sempre na mesma ordem. Cada servidor foi submetido a mesma carga de trabalho para medir o impacto da política de escalonamento utilizada nos tempos de resposta e execução de cada *quest*. Após concluir as *quests* de uma aplicação, a configuração dos servidores da rede foi trocada, e os mesmos testes foram repetidos.

A aplicação de testes contém duas baterias de testes, uma com apenas aplicações determinísticas, e outra com aplicações não-determinísticas, descritas nas próximas seções. Foi esperada a conclusão dos testes da bateria determinística antes da execução dos da bateria não-determinística. Os arquivos de LOG de cada servidor e do cliente foram analisados para obter os resultados apresentados neste artigo, que são a média geométrica das três repetições de cada bateria de testes.

4.2 Aplicações Determinísticas

Foram utilizadas duas aplicações determinísticas, cujo resposta, para uma dada entrada, é sempre a mesma. Uma aplicação é o cálculo do *hash* SHA-256 de quatro arquivos diferentes, seis mil vezes. Cada um dos quatro arquivos tem um tamanho diferente, e contém valores binários quaisquer. Os tamanhos de arquivo utilizados foram 1Kb, 8Kb, 16Kb e 128Kb. A outra aplicação testada foi uma compilação de códigos fonte na linguagem "C", utilizando o compilador "Gnu C/C++ Compiler". A compilação é uma operação de maior tempo de execução do que o cálculo do *hash* dos arquivos testados.

4.3 Aplicações Não-Determinísticas

Aplicações não determinísticas podem, para os mesmos dados de entrada, produzir resultados distintos em cada execução. Isso se deve a natureza randômica e adaptativa deste tipo de aplicação. As aplicações não-determinísticas utilizadas são funções do projeto WEKA, que implementa algoritmos de inteligência artificial na linguagem JAVA, comprimidos em um único arquivo executável JAR (HALL et al., 2009). As funções utilizadas foram selecionadas em parte por terem alta intensidade computacional, e por apresentarem um comportamento não-determinístico, não sendo possível determinar previamente o tempo exato da conclusão de sua execução, mas sendo possível dizer quais são executadas mais rapidamente.

As funções selecionadas se encaixam na categoria dos *classificadores*, algoritmos que decidem a qual das categorias previamente estabelecidas pertence um novo dado observado (WITTEN; FRANK, 2005).

Cada função utiliza um algoritmo distinto, e foi testada com duas bases de dados distintas, fornecidas juntamente com o sistema WEKA como exemplos. O Quadro 4.3 apresenta as funções testadas, ordenadas de forma crescente pelo seu tempo médio de execução.

Quadro 1 - Algoritmos Classificadores Testados
Por ordem crescente de tempo médio de execução

Classificador	Base de Dados	
Árvore J48	weather.numeric.arff	weather.nominal.arff
Decision Stump	cpu.with.vendor.arff	cpu.arff
Naive Bayes	credit-g.arff	iris.2D.arff
Ada Boost M1	breast-cancer.arff	labor.arff
SMO	breast-cancer.arff	labor.arff

Fonte: Dados da pesquisa

5 RESULTADOS EXPERIMENTAIS

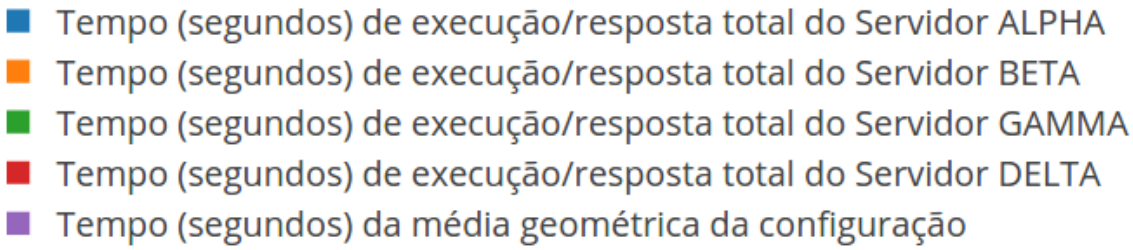
As Figuras 7 e 8 mostram, respectivamente, por configuração, os tempos de resposta e execução para as aplicações determinísticas, e as Figuras 9 e 10, para as aplicações não-determinísticas. A Figura 6 mostra a legenda para todas as demais.

Cada configuração contém um conjunto de cinco barras verticais, sendo a quinta barra o valor da média geométrica das demais. A média geométrica foi utilizada no lugar da aritmética porque é uma medida menos afetada por poucos resultados excêntricos em um conjunto de outra forma homogêneo, expressando com maior precisão a informação representada por um conjunto de valores. Cada uma das primeiras quatro barras representa o tempo medido em um servidor, ordenados pelo alfabeto grego. O Servidor ALPHA corresponde ao servidor de menor endereço de IP, o servidor BETA ao segundo menor, e assim por diante. Assim, na configuração 202, as primeiras duas barras correspondem ao tempo medido nos servidores ALPHA e BETA, que utilizavam o escalonamento sem limite de processos concorrentes. As duas barras seguintes representam o tempo medido nos dois servidores utilizando o escalonamento SJF. A quinta barra mostra a média geométrica das demais.

Os resultados dos testes revelam que uma tarefa submetida à grade Assistance passa muito mais tempo em espera do que em execução, dada a diferença entre estes valores nos gráficos apresentados, em especial os das figuras 8 e 7, onde a diferença alcança uma escala de grandeza. Observa-se variância no tempos de execução e processamento em cada servidor, independentemente da configuração da rede ou do algoritmo de escalonamento usado no servidor. Aplicações determinísticas têm uma maior relação de tempo de execução sobre o tempo de resposta (relação E/R) do que aplicações não-determinísticas. A relação E/R revela o quanto a execução distribuída e paralela é mais rápida do que a execução serial das tarefas.

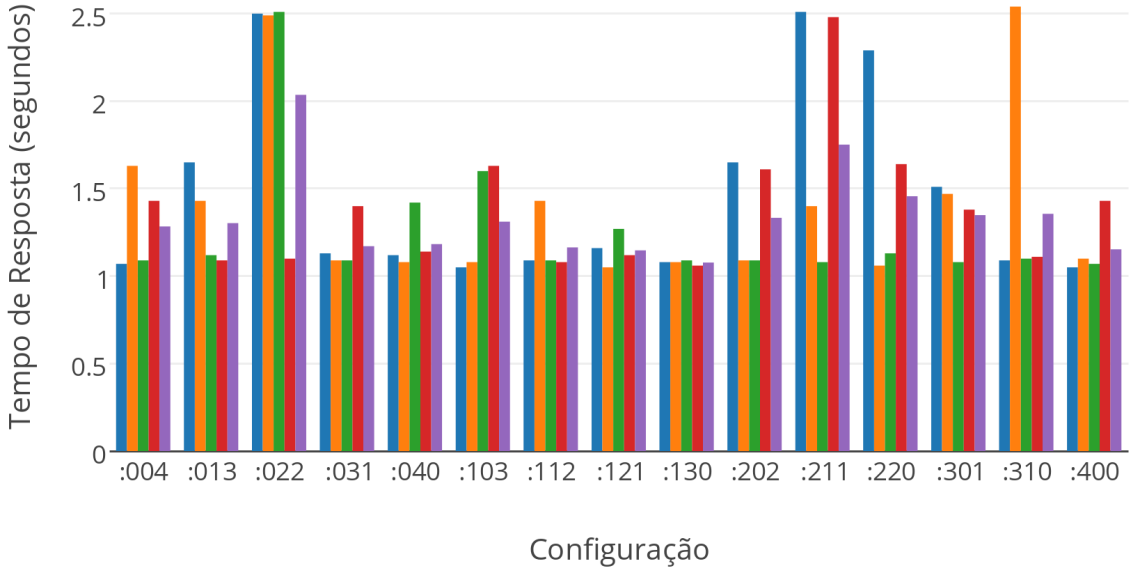
A configuração que apresentou os menores tempos de espera, nos dois tipos te aplicações, foi a configuração 031, que também tem o menor tempo de execução para aplicações não-determinísticas. Observa-se que algumas configurações que utilizam mais de uma política de escalonamento simultaneamente, como 031 e 220, apresentam tempo de resposta menor do que o das configurações 004 e 040, que se concentram em um único algoritmo, respectivamente, SJF e FIFO.

Figura 6 – Legenda para os Gráficos das Figuras desta Seção



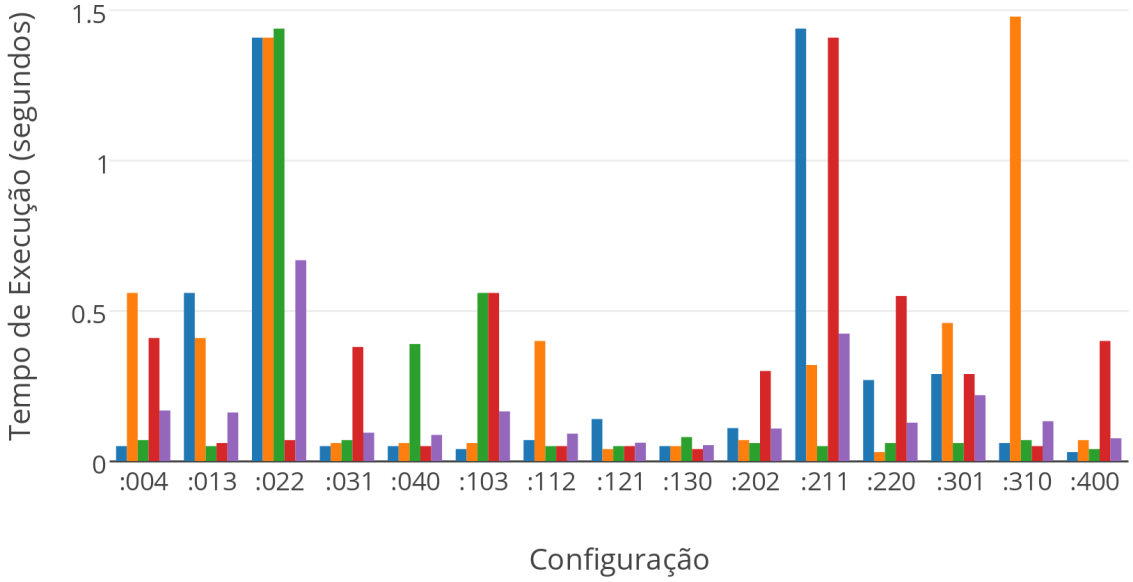
Fonte: Elaborado pelo autor.

Figura 7 – Tempo de Resposta das Aplicações Determinísticas por Configuração



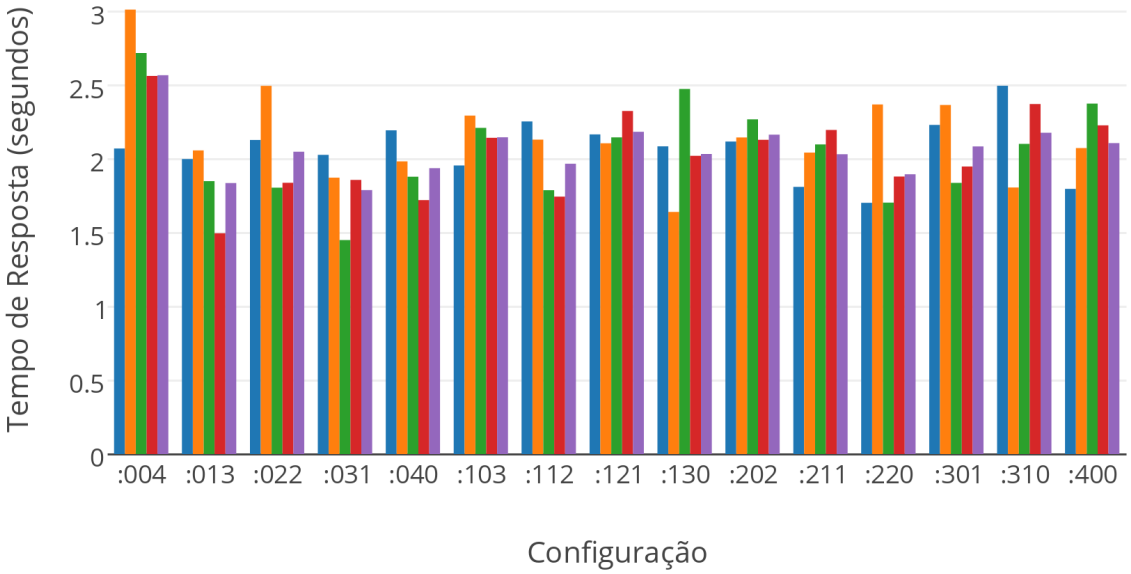
Fonte: Dados da pesquisa

Figura 8 – Tempo de Execução das Aplicações Determinísticas por Configuração



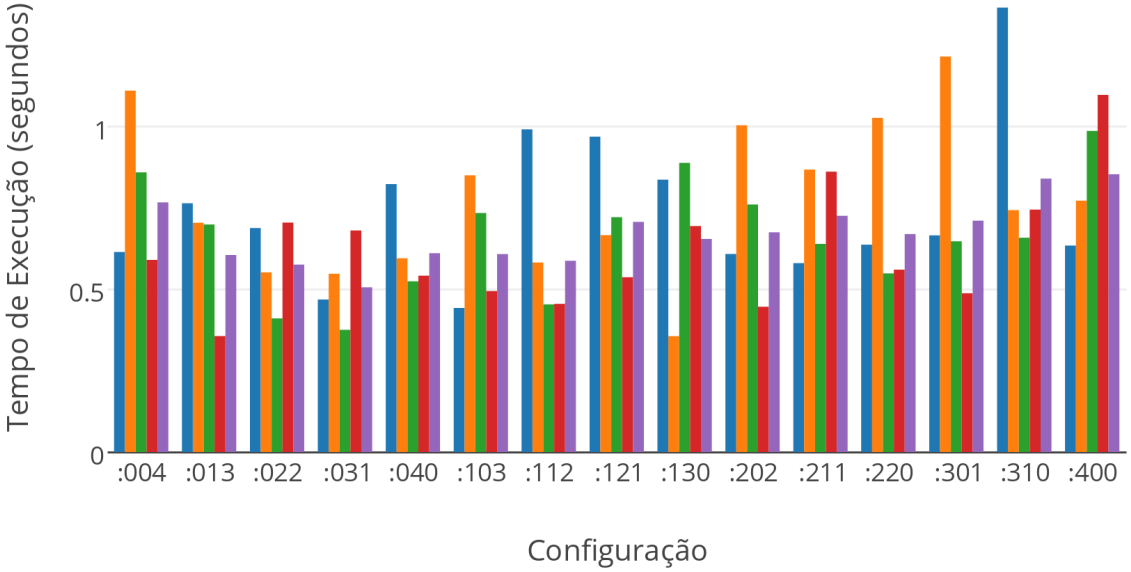
Fonte: Dados da pesquisa

Figura 9 – Tempo de Resposta das Aplicações Não-Determinísticas por Configuração



Fonte: Dados da pesquisa

Figura 10 – Tempo de Execução das Aplicações Não-Determinísticas por Configuração



Fonte: Dados da pesquisa

Seja o tempo de execução local (TEL) o tempo que seria gasto para executar todos os testes realizados com uma mesma configuração, serialmente, em um único dispositivo similar aos utilizados nos testes. O TEL pode ser calculado pela soma dos tempos de execução de todos os testes realizados com uma dada configuração, uma vez que, realizados localmente, não têm necessidade de tráfego de dados pela rede. O tempo de resposta da grade (TRG) é o tempo que a grade Assistance, em uma determinada configuração, demora para completar todas as tarefas submetidas. Tal tempo pode ser considerado como o tempo de resposta máximo de uma dada configuração.

A Tabela 1 mostra, para cada configuração e tipo de aplicações, o TEL, o TRG e a relação E/R da mesma. Uma relação E/R superior à 100% indica que o tempo de execução serial

Tabela 1 – Tempos de Resposta, Execução e Relações E/R

	Aplic. Não-Determinísticas			Aplic. Determinísticas		
Configuração	**TRG(s)**	**TEL(s)**	**E/R**	**TRG(s)**	**TEL(s)**	**E/R**
Conf. 004	3,01	3,94	130,80%	1,63	1,26	77,20%
Conf. 013	2,06	3,13	152,00%	1,65	1,24	75,27%
Conf. 022	2,50	2,93	117,44%	2,51	5,00	199,17%
Conf. 031	2,03	2,58	127,13%	1,40	0,65	46,75%
Conf. 040	2,20	3,10	141,01%	1,42	0,64	44,89%
Conf. 103	2,30	3,13	136,40%	1,63	1,38	85,01%
Conf. 112	2,26	3,07	136,12%	1,43	0,66	46,26%
Conf. 121	2,33	3,60	154,79%	1,27	0,34	26,86%
Conf. 130	2,48	3,43	138,58%	1,09	0,27	25,06%
Conf. 202	2,27	3,50	153,95%	1,65	0,65	39,30%
Conf. 211	2,20	3,68	167,24%	2,51	3,64	145,20%
Conf. 220	2,37	3,44	145,24%	2,29	1,04	45,32%
Conf. 301	2,37	3,73	157,47%	1,51	1,32	87,38%
Conf. 310	2,50	4,35	174,25%	2,54	1,79	70,58%
Conf. 400	2,38	4,35	182,76%	1,43	0,62	43,09%

Fonte: Dados da pesquisa

das tarefas executadas em uma configuração seria maior do que o tempo de resposta da grade para as mesmas tarefas. Sendo X a relação E/R de uma configuração, o ganho proporcional de desempenho da grade sobre a execução serial é dado por X-100. Pode-se ver que o uso do sistema Assistance torna o processamento dos testes de aplicações não-determinísticas em média 47,68% mais rápido, sendo que as aplicações requerem, em média geométrica, 0,67 segundos para serem executadas. A mesma tabela informa que o uso do sistema Assistance torna o processamento dos testes determinísticos mais lento, com a relação E/R abaixo de 100%. Isso se deve à latência da rede, que acarreta sobrecarga no uso de um sistema de grades computacionais, e ao fato de as aplicações determinísticas utilizadas serem completadas muito rapidamente, numa média geométrica de 0,14 segundos, tornando a latência da rede o fator determinante de seu tempo de resposta.

6 CONCLUSÃO

Os resultados obtidos nos testes revelam que o Assistance oferece benefícios característicos de um sistema de computação em grade oportunística com escalonamento distribuído. Os resultados também revelam que mais pesquisas serão necessárias para explicar o comportamento do sistema nos casos em que diferentes políticas de escalonamento foram utilizadas simultaneamente na mesma grade. É possível dizer que o sistema apresenta uma sobrecarga que afeta o tempo de execução de tarefas no mesmo, sendo necessário um aperfeiçoamento da implementação do sistema para tornar o *middleware* mais eficiente.

Os trabalhos relacionados afirmam a necessidade de um *middleware* capaz de operar entre diferentes sistemas de grade, e que ofereça mais flexibilidade e segurança aos proprietários

de recursos de uma grade. O Assistance permite que implementações distintas, mesmo fora do escopo do sistema, interajam com o mesmo, utilizando quaisquer otimizações. A arquitetura do Assistance, que evita a passagem de códigos de um dispositivo a outro, também evita que o sistema seja usado como porta de entrada de códigos maliciosos. Assim, conclui-se que o Assistance oferece uma solução para os desafios da interoperabilidade, principalmente flexibilidade e segurança em grades computacionais oportunísticas.

Os trabalhos futuros podem avaliar o desempenho ao escalar o sistema Assistance, para envolver mais servidores e clientes, requisitando uma quantidade maior de tarefas. Também pode-se testar o desempenho do sistema com aplicações diferentes, em uma rede heterogênea ou de maior alcance, como a Internet, pode revelar o benefício que o mesmo representa caso executado em uma rede de maior latência. Além disso, testes em maior escala podem ajudar a explicar o comportamento do sistema em configurações que utilizam diversos algoritmos de escalonamento. Melhoramentos podem ser feitos no protocolo de comunicação do Assistance, para lidar com perda e corrupção de dados em pacotes, e a latência da rede. Por último, testar o desempenho do Assistance numa grade com dispositivos heterogêneos, escalonando tarefas de acordo com as capacidades de cada dispositivo, pode também revelar mais informações sobre o comportamento do sistema, além de diferentes políticas de escalonamento.

Referências

ANDERSON, David P. Public computing: Reconnecting people to science. In: **Conference on Shared Knowledge and the Web**. [S.l.: s.n.], 2003. p. 17–19.

ANDERSON, David P. Boinc: A system for public-resource computing and storage. In: IEEE. **Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on**. [S.l.], 2004. p. 4–10.

CAI, Xing; LANGTANGEN, Hans Petter; MOE, Halvard. On the performance of the python programming language for serial and parallel scientific computations. **Scientific Programming**, Hindawi Publishing Corporation, v. 13, n. 1, p. 31–56, 2005.

CEDERSTRÖM, Andreas. **On using Desktop Grid Computing in software industry**. 2010. Dissertação (Mestrado) — School of Engineering, Blekinge Institute of Technology, Ronneby, Suécia.

CIRNE, Walfredo et al. Labs of the world, unite!!! **Journal of Grid Computing**, Springer, v. 4, n. 3, p. 225–246, 2006.

FEDAK, Gilles et al. Xtremweb: A generic global computing system. In: IEEE. **Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on**. [S.l.], 2001. p. 582–587.

FOSTER, Ian; IAMNITCHI, Adriana. On death, taxes, and the convergence of peer-to-peer and grid computing. In: **Peer-to-Peer Systems II**. [S.l.]: Springer, 2003. p. 118–128.

FOSTER, Ian; KESSELMAN, Carl. Globus: A metacomputing infrastructure toolkit. **International Journal of High Performance Computing Applications**, SAGE Publications, v. 11, n. 2, p. 115–128, 1997.

FOSTER, Ian; KESSELMAN, Carl. The grid in a nutshell. In: **Grid resource management**. [S.l.]: Springer, 2004. p. 3–13.

FOSTER, Ian; KESSELMAN, Carl. The history of the grid. **computing**, v. 20, n. 21, p. 22, 2010.

FOSTER, Ian; KESSELMAN, Carl; TUECKE, Steven. The anatomy of the grid: Enabling scalable virtual organizations. **International journal of high performance computing applications**, Sage Publications, v. 15, n. 3, p. 200–222, 2001.

FOSTER, Ian; KESSELMAN, C; TUECKE, S. **What is the grid?-a three point checklist. GRIDtoday,(6), July 2002**. 2002.

FOSTER, Ian et al. Cloud computing and grid computing 360-degree compared. In: IEEE. **Grid Computing Environments Workshop, 2008. GCE'08**. [S.l.], 2008. p. 1–10.

GOSLING, James; MCGILTON, Henry. The java language environment: A white paper, 1995. **Sun Microsystems**, v. 85, 2003.

HALL, Mark et al. The weka data mining software: an update. **ACM SIGKDD explorations newsletter**, ACM, v. 11, n. 1, p. 10–18, 2009.

KAHANWAL, Dr; SINGH, Dr TP et al. The distributed computing paradigms: P2p, grid, cluster, cloud, and jungle. **arXiv preprint arXiv:1311.3070**, 2013.

KSHEMKALYANI, Ajay D; SINGHAL, Mukesh. **Distributed computing: principles, algorithms, and systems**. [S.l.]: Cambridge University Press, 2008.

LITZKOW, Michael J; LIVNY, Miron; MUTKA, Matt W. Condor-a hunter of idle workstations. In: IEEE. **Distributed Computing Systems, 1988., 8th International Conference on**. [S.l.], 1988. p. 104–111.

MCCOOL, Michael; REINDERS, James; ROBISON, Arch. **Structured parallel programming: patterns for efficient computation**. [S.l.]: Elsevier, 2012.

ROSSUM, Guido Van; DRAKE, Fred L. Python tutorial, release 2.2. 1. Citeseer, 2002.

SCHWIEGELSHOHN, Uwe et al. Perspectives on grid computing. **Future Generation Computer Systems**, Elsevier, v. 26, n. 8, p. 1104–1115, 2010.

SHARMA, Dipti; MITTAL, Pradeep. Job scheduling algorithm for computational grid in grid computing environment. **International Journal of Advanced Research in Computer Science and Software Engineering**, 2013.

SILVA, Daniel Paranhos Da; CIRNE, Walfredo; BRASILEIRO, Francisco Vilar. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In: **Euro-Par 2003 Parallel Processing**. [S.l.]: Springer, 2003. p. 169–180.

SNAVELY, Allan et al. Benchmarks for grid computing: a review of ongoing efforts and future directions. **ACM SIGMETRICS Performance Evaluation Review**, ACM, v. 30, n. 4, p. 27–32, 2003.

SUBRAMANI, Vijay et al. Distributed job scheduling on computational grids using multiple simultaneous requests. In: IEEE. **High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on**. [S.l.], 2002. p. 359–366.

TANENBAUM, Andrew S. Modern operating systems. Pearson Education, 2009.

TANENBAUM, Andrew S. **Computer networks**. 5. ed. [S.l.]: Prentice Hall PTR, 2011.

THAIN, Douglas; TANNENBAUM, Todd; LIVNY, Miron. Distributed computing in practice: The condor experience. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, v. 17, n. 2-4, p. 323–356, 2005.

VAQUERO, Luis M et al. A break in the clouds: towards a cloud definition. **ACM SIGCOMM Computer Communication Review**, ACM, v. 39, n. 1, p. 50–55, 2008.

WIKIPEDIA. **Kinect — Wikipedia, The Free Encyclopedia**. 2015. [Online; accessed 25-June-2015]. Disponível em: <https://en.wikipedia.org/w/index.php?title=Kinect&oldid=667885564>.

WITTEN, Ian H; FRANK, Eibe. **Data Mining: Practical machine learning tools and techniques**. [S.l.]: Morgan Kaufmann, 2005.

XHAFA, Fatos; ABRAHAM, Ajith. Computational models and heuristic methods for grid scheduling problems. **Future generation computer systems**, Elsevier, v. 26, n. 4, p. 608–621, 2010.

ZHANG, Qi; CHENG, Lu; BOUTABA, Raouf. Cloud computing: state-of-the-art and research challenges. **Journal of internet services and applications**, Springer, v. 1, n. 1, p. 7–18, 2010.

ZHAO, Han; LIU, Xinxin; LI, Xiaolin. A taxonomy of peer-to-peer desktop grid paradigms. **Cluster Computing**, Springer, v. 14, n. 2, p. 129–144, 2011.