Uma Arquitetura de Referência para Plataforma de Crowdsensing em Smart Cities

Alternative Title: A Reference Architecture for a Crowdsensing Platform in Smart Cities

Herbertt B. M. Diniz
Centro de Informática, UFPE
Jornalista Aníbal Fernandes
Recife, Brazil 50.740-560
hbmd@cin.ufpe.br

Emanoel C. G. F. Silva
Centro de Informática, UFPE
Jornalista Aníbal Fernandes
Recife, Brazil 50.740-560
ecgfs@cin.ufpe.br

Kiev Santos da Gama
Centro de Informática, UFPE
Jornalista Aníbal Fernandes
Recife, Brazil 50.740-560
kiev@cin.ufpe.br

RESUMO

Em decorrência dos problemas gerados pelo crescimento populacional nas grandes cidades, surge a necessidade de soluções que apontem para a iniciativa de "Cidades Inteligentes" (*Smart Cities*), ou seja, que utilizam a tecnologia para oferecer recursos, que podem ajudar a solucionar ou minimizar os problemas urbanos. Essas soluções buscam a integração de diversas fontes de Tecnologias da Informação (TICs), no entanto, essas fontes de Tecnologias da Informação formam estruturas complexas e geram um grande volume de dados, que apresentam enormes desafios e oportunidades, dificultando a possibilidade de se disponibilizar programas que integrem informações de sensores e que captem o espaço físico, tirando amostras do que está acontecendo em tempo real. Com intuito de fornecer uma arquitetura de referência, a fim de confrontar questionamentos acerca desses desafios e oportunidades, neste artigo é apresentada uma abordagem, utilizando componentes *off-the-shelves* (componentes de prateleira), uma plataforma de *Crowdsensing* para solução de *smart cities*, também foi realizado um experimento para determinar a resistência e a estabilidade do sistema. Assim, esta proposta abre caminho para que se amplie a integração de dados de fontes dos mais variados tipos de sensores e dispositivos.

Palavras-Chave

Cidades Inteligentes, CEP, Arquitetura, Sensores Humanos

ABSTRACT

Due to the problems caused by population growth in large cities, there is a need for solutions that point to the initiative of Smart Cities, that is, using the technology to offer resources that can help solve or minimize urban problems. This solutions seeks the integration of several sources of Information Technology(ICTs), however, those source of Information Technology form complex structures and generate a large volume of data, that represents big challenges and opportunities, impeding the possibility of to make available, systems that integrate informations of sensors and capture data from the physical space, getting samples of what is going on in the city in real time. With the intention of offer a reference architecture, to compare issues related with those challenges and opportunities, this paper presents an approach that employs components off-the-shelves for the construction of a crowdsensing platform for solution in Smart Cities. We performed an experiment to determine the performance and stability of the system. Thus, this proposal opens the way for to broaden the integration of data sources of various types of sensors and devices.

Categorias e Descritores do Assunto

D.2.11 [**Software Engineering**]: Software Architectures; D.2.12 [**Software Engineering**]: Interoperability

Termos Gerais

Performance Standardization

Keywords

Smart Cities, CEP, Architecture, Crowdsensing

1. INTRODUÇÃO

Em consequência da evolução tecnológica, vários dispositivos eletrônicos, informatizados, estão se integrando através de redes de internet, se incorporando a todos os espaços e ambientes das cidades. Isso permite a utilização de tecnologias para monitoramento urbano[20], influenciando a nossa maneira de interagir com os espaços urbanos os quais estão cada vez mais caóticos, em decorrência do inchaço populacional.

Dados estatísticos demonstram que há um enorme crescimento populacional nas grandes cidades, que tende a aumentar de forma alarmante nos próximos anos[6], acarretando assim, enormes desafios na solução de problemas legados a vários domínios específicos, principalmente, no que diz respeito à mobilidade urbana, segurança, educação, utilização eficiente dos recursos naturais e descarte de resíduos, sem que haja prejuízos para o meio ambiente.

Em decorrência desses fatos, surge a necessidade de soluções que convirjam para iniciativa de "Cidades Inteligentes" (*smart cities*), ou seja, aquelas que utilizam a tecnologia para oferecer recursos que podem ajudar a solucionar ou minimizar os problemas urbanos em seus domínios específicos. Cidades inteligentes[19] é um termo que ganhou força na academia, empresas e governo para descrever cidades que, por um lado, são cada vez mais compostas e monitoradas por computação pervasiva e ubíqua e, por outro, cuja economia e governança está sendo impulsionada pela inovação, criatividade e empreendedorismo, interpretados por especialistas.

Segundo Naphade *et al.*[19], várias iniciativas já estão em curso nesse sentido, a exemplo do modelo de sustentabilidade aplicado em Dubuque no estado americano Iowa, em Seul na Coreia do Sul e do Centro de Operações da prefeitura do Rio de Janeiro. Elas buscam através da integração de diversas fontes de Tecnologias da Informação (TICs), tais como Internet das Coisas (IoT), Cloud Computing, Sistemas de Informações Geográficas (SIGs) e Dispositivos Móveis, além da utilização dos próprios cidadãos, levando em consideração uma visão holística da cidade e centralizando o monitoramento e controle para auxiliar no apoio à tomada de decisão [15].

No entanto, essas fontes de TICs formam estruturas complexas[18], e geram um grande volume de dados, que apresentam enormes desafios e oportunidades, dificultando a possibilidade de se disponibilizar programas que integrem informações de sensores e que captem o espaço físico, tirando amostras do que está acontecendo em tempo real. A maioria das soluções existentes nesse sentido são proprietárias e possuem alto custo, a despeito disso o uso combinado de FLOSS (*Freely Licensed Open Source Software*)[22] e COTS (*Commercial Off-The-Shelf*)[9] possibilita a criação de soluções de baixo custo e com pouco esforço de implementação, segundo Gasperoni[16] o uso de FLOSS e COTS oferece inúmeras vantagens, a exemplo da qualidade de software e rapidez na evolução.

Com intuito de fornecer uma arquitetura de referência, a fim de confrontar questionamentos acerca desses desafios e oportunidades, neste artigo é apresentada uma abordagem, utilizando componentes *off-the-shelves* (componentes de prateleira), que atende e implementa de forma eficaz e simplificada uma plataforma de *middleware* para solução de *smart cities*, configurável para vários domínios, utilizando Processamento de Eventos Complexos, para análise de streams de dados em tempo real e integrando sensores através de serviços web e fila de mensagem assíncrona.

Foram conduzidos alguns experimentos para provar a viabilidade da abordagem proposta. Assim, foram realizados testes para avaliar a capacidade de resposta às requisições, eficiência do tempo de resposta e o comportamento do serviço assíncrono de fila de mensagem. O objetivo dos testes foi determinar a resistência e a estabilidade do sistema.

O presente trabalho está organizado da seguinte forma: na Seção 2, são apresentados trabalhos relacionados; na Seção 3, é apresentada a visão geral e descrição do sistema; na Seção 4, são apresentados as ferramentas e os métodos dos testes realizados na Seção 5. Na Seção 6, são apresentados resultados e discussões e, por fim, na Seção 7, são apresentadas conclusões e trabalhos futuros.

2. TRABALHOS RELACIONADOS

Muitas pesquisas relacionadas ao gerenciamento e integração de sensores heterogêneos em espaços públicos, estão baseadas em *mobile crowd sensoring* e EDA (*Event Driven Architecture*). Segundo Etzion e Niblett[12], EDA é um estilo arquitetural em que um ou mais componentes de software respondem ao receber um ou mais eventos de notificação. *Mobile Crowdsensing*[15], por sua vez, refere-se a sensoriamento comunitário (*i.e.*, através de pessoas) para coleta de dados, no qual as pessoas funcionam como sensores (*crowd sensors*). Abaixo é apresentada uma lista não exaustiva de plataformas construídas utilizando as abordagens citadas.

SOFIA[14] é uma plataforma de interoperabilidade semântica entre um conjunto selecionado de aplicações e um ecossistema entre objetos interativos, como sensores, atuadores e outros sistemas embarcados. Seu objetivo é oferecer recursos para a construção de ambientes inteligentes. Alguns requisitos destes ambientes inteligentes são o armazenamento de informação e procedimentos de busca a sistemas embarcados, implementadas em diferentes tecnologias. O projeto aborda três áreas de aplicação que representam diferentes tipos de espaço em termos de escala, aplicações e serviços: espaços pessoais inteligentes (por exemplo, automóveis), espaços interiores inteligentes (por exemplo, casa e escritório), e espaços da cidade inteligentes (por exemplo, infra-estrutura alargada e facilidades como uma estação de metrô, shopping e assim por diante).

Borja e Gama[8] propõem uma plataforma baseada em barramento de serviços com o objetivo de integrar diferentes fontes de dados em diferentes sistemas e de dispositivos. Por exemplo prático, a plataforma é aplicada ao domínio de monitoramento do sistema de transporte público da cidade do Recife. Como resultado, foi demonstrado que aplicações de terceiros podem ser construídas a partir dos dados coletados destas fontes de dados heterogêneas.

TrafficInfo[13] é um aplicativo baseado em *Crowd sensors*, que visualiza em tempo real as informações de transporte público da cidade pelo Google Maps. Ele é construído utilizando modelo *publish/subscribe* para comunicação dos passageiros que se inscrevem no TrafficInfo, de acordo com seu interesse, para os canais de informação de trânsito dedicadas a diferentes linhas de transportes públicos ou pontos de paradas. Assim, eles são informados sobre a situação do transporte público ao vivo, como por exemplo, as posições reais dos veículos. TrafficInfo basicamente visualiza a circulação de veículos de transporte público em um mapa. Quando os dados em tempo real são recolhidos pelos passageiros, a informação da posição adquirida é atualizada refletindo a situação do tráfego. Assim, os passageiros podem ter uma ideia se o veículo selecionado está na hora ou atrasado, se eles tem que se apressarem ou procurarem algum transporte alternativo.

CrowdOut[7] é também um serviço de *crowd sensing* móvel para cidades inteligentes, que permite aos usuários relatar problemas de segurança rodoviária que eles experimentam em seu ambiente urbano (excesso de velocidade, carros estacionados ilegalmente, sinais e símbolos, qualidade de estradas e níveis de tráfego, etc.) e partilhar esta informação com a sua comunidade. Basicamente, um usuário pode especificar o tipo de crime, tirar uma foto com seu smartphone e adicioná-lo ao relatório para apresentá-la; ele também pode adicionar um comentário curto (140 caracteres, como no Twitter) para descrever a infração de forma mais precisa. O serviço também registra as coordenadas

GPS dos usuários, a fim de indicar com precisão o local do delito ocorrido na estrada. Em seguida, os relatórios são exibidos num mapa em tempo real.

Choi & Kang[10] propõem uma plataforma em HTML5, com serviço baseado em localização, fazendo uso do GPS, a fim de achar pontos de interesse nas proximidades. Como resultado, apresentam um case de uma ferramenta de dispositivo móvel, para ajudar estrangeiros e coreanos num complicado processo de registro de aquisição e alienação de imóveis na Coreia do Sul. Neste trabalho foi desenvolvido um aplicativo móvel utilizando o framework de aplicação Sencha Touch[5] utilizando serviços RESTful através do framework Restlet[4].

Ao analisar estas soluções, percebe-se que a solução proposta neste trabalho se diferencia da plataforma SOFIA por possuir um escopo mais fechado, objetivando oferecer serviços de *Crowdsensing*. Em contrapartida, a apresentada pela SOFIA aborda várias áreas de aplicações. Apesar disso, nossa proposta permite a configuração de domínio, contrastando nesse aspecto com as plataformas TrackInfo e a proposta apresentada por Choi & Kang. A plataforma CrowdOut, contrasta com a nossa por não possuir recurso de informação de alertas disparados através de processamento de eventos complexos. Já Borja e Gama propõem uma solução utilizando arquitetura de barramento que, pelo fato de oferecer recursos além do necessário para fornecer os serviços propostos, torna a solução mais complexa e difícil de administrar.

3. VISÃO GERAL E DESCRIÇÃO DO SISTEMA

3.1 Visão Geral

Diante das soluções de *middleware* para *smart cities* propostas mais comumente atualmente, algumas características foram levadas em consideração na construção de nossa arquitetura, por serem tratadas de forma comum nas mais diferentes abordagens. A exemplo disso, pode-se citar o uso de conexão de sensores usando serviços de fila de mensagem para a integração de fontes de dados de maneira assíncrona e síncrona utilizando serviço web restful, processamento de eventos complexos para análise de *streams* de dados em tempo real e uso de sensores humanos através de sistemas de *Crowdsensing*. Para tal, utilizamos alguns componentes de prateleira, com intuito de facilitar e simplificar a implementação. O papel desempenhado por cada componente será apresentado mais adiante na descrição da arquitetura da plataforma, a qual foi construída baseada nessas características.

O projeto arquitetural da plataforma é descrito em 2 módulos de serviços: *Front-End Module* e *Back-End Module*, conforme mostra a Figura 1. No Módulo *Front-End* (Figura 1 (1)), encontra-se o sistema de *Crowdsensing*, no módulo de Back-End(Figura 1 (2)), a API de serviços restful(Figura 1 (2.*a*)), que coordena o acesso aos Serviços Síncronos de *Request/Response* e ao serviço de Processamento Eventos(Figura 1 (2.*c*)). Este, por sua vez, acessa os serviços do módulo serviço de mensagens assíncronas no modelo *Publish/Subscribe* (Figura 1 (2.*d*)), disponibilizando assim a transmissão de dados em tempo real e a sinalização de dispositivos, que possibilita estabelecer e manter conexões via Socket e Restful, permitindo qualquer dispositivo enviar e receber dados para serem processados e redistribuídos. Estes dados são controlados pelo Gerenciador de Serviços da aplicação, (Figura 1 (2.*b*)), que acessa através do serviço de persistência, os dados persistidos na base de Dados (Figura 1 (2.*e*)).

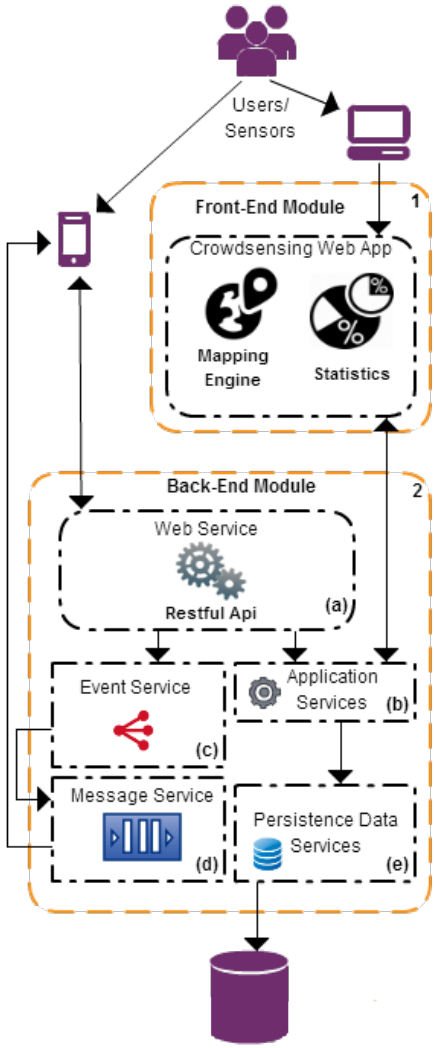

Figura 1: Arquitetura da plataforma

3.2 Descrição do Sistema

A Plataforma possui dois serviços de entrada de dados, que podem ser providos através de Sistema de *Crowdsensing* no módulo *Front-End* ou através da API Restful do *Web Service*no módulo *Back-End*. A seguir descrevemos respectivamente as duas abordagens.

3.2.1 Sistema de Crowdsensing

A plataforma proposta possui um recurso que permite a participação colaborativa dos próprios usuários, possibilitando a captação da percepção dos mesmos sobre o ambiente em que vivem, tornando possível a inserção das pessoas como elementos mais participativos no sistema e na cidade, transformando-os em sensores humanos (*Crowd Sensors*). Segundo Erickson[11], "humanos podem contribuir com co-

nhecimento qualitativo mais profundo, analisar dados vagos ou incompletos e agir de uma forma em que sistemas digitais frequentemente não são capazes". Nesse âmbito foi desenvolvido um Sistema de *Crowdsensing*, que possui um serviço do Google Maps, o qual permite aos usuários na forma de sensores humanos visualizar pontos de ocorrências no mapa através de marcadores, os gráficos com estatísticas e a tabela com a lista de detalhada com dados de ocorrências.

Segundo Sherchan *et al.*[21], há duas categorias de *crowdsensing*: *personal* e *community sensing* (sensoriamento pessoal e comunitário), onde o sensoriamento pessoal refere-se ao tratamento de dados de um único indivíduo para beneficiar a este mesmo indivíduo com o resultado do processamento dos dados por ele emitido. Em se tratando de sensoriamento comunitário, [15] existem mais dois tipos de modelos distintos: o sensoriamento oportunista, que requer um envolvimento mínimo do usuário (ex: localização automática por GPS), e o sensoriamento participativo, que requer a participação ativa e direta no fornecimento de dados dos usuários.

Na plataforma proposta, a aplicação de *crowdsensing* é parametrizável para vários domínos oferecendo a oportunidade de criar marcadores customizáveis, a exemplo do mapa descrito na Figura 2, com dados do domínio de mobilidade urbana, descrevendo pontos de acidentes de trânsito ocorridos com bicicletas. Essa ferramenta se enquadra na categoria de *community sensing* e no modelo de sensoriamento participativo, por ser voltada para o controle de fenômenos de grande escala, e de muitos indivíduos participantes e pela participação direta dos usuários.

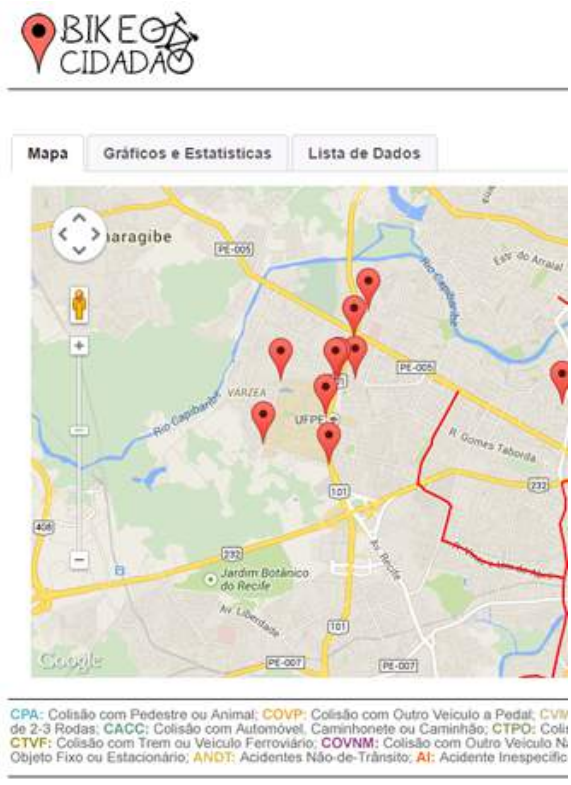

Figura 2: Tela principal do sistema de *crowdSensing*

A ferramenta web ainda possui um painel com gráficos e estatísticas que apresenta a totalização de ocorrências, um gráfico em formato pizza que mostra a estatística de cada tipo de ocorrência, e a lista de ocorrências em formato tabular.

3.2.2 *Services*

Segundo Maréchaux[17], aplicações de negócios precisam ser conectadas a fim de criar uma solução integrada. Arquitetura Orientada a Serviços (*Service-oriented Architecture*-SOA) e arquitetura orientada a eventos (*Event Driven Architecture*-EDA) são dois paradigmas diferentes que abordam os desafios da integração complexos. Essas aplicações tem como características principais flexibilidade e responsividade. O desafio de TI tem sido apoiar esta visão de negócios com as arquiteturas e tecnologias apropriadas. O modelo a ser descrito a seguir é uma proposta simplificada baseada na utilização de EDA e SOA através do uso de alguns componentes de prateleira.

O modelo de *Web Services*, apresentado na Figura 3, provê serviços de dados em tempo real através de requisições síncronas e assíncronas. As requisições síncronas são feitas através de web service RESTful utilizando JSON como padrão de transmissão de dados. As requisições via REST são realizadas através do envio de dados do cliente solicitando serviços providos pela camada *Web Service* via POST/GET que, por sua vez, solicita dados da persistência através de funções parametrizáveis. Como exemplo, tem-se o serviço de localização que retorna o tipo de ocorrência e a distância entre um ponto de latitude e longitude enviado pelo cliente e a localização do ponto de ocorrência mais próximo, persistido pelo usuário(sensor) através do serviço de *Crowdsensing*, descrito no Item 3.2.1

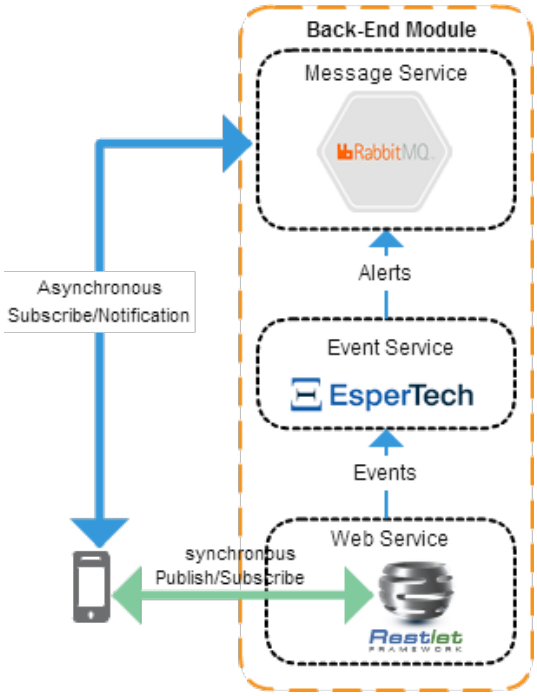

Figura 3: Modelo da camada *Service*

À medida que as requisições chegam, eventos são registrados no sistema CEP Esper[2] e alertas são disparados para

filas no serviço de processamento de eventos do modelo assíncrono para serem consumidos pelos clientes. Desse modo, esse modelo utiliza o serviço de mensagens em fila, para capturar os alertas de CEP disparados de maneira assíncrona, através da captura de eventos e processamento das regras, além de fornecer serviços através do framework Restlet, processando ocorrências armazenadas na camada de dados, enviadas pelos sensores humanos.

O Módulo de *Web Services* provê serviços de localização que podem ser parametrizáveis através das funções e das customizações das regras da ferramenta de CEP, além de possibilitar o usuário a criação de novas regras para serem providas através do serviço de filas do sistema RabbitMQ[3], no modelo assíncrono. Uma das regras desse modelo (ilustrada pelo Listing 1) é de localização parametrizável, que permite disparar alertas com a distância, em tempo real, das ultimas duas requisições de localização de pontos de latitude e longitude enviados através do serviço de REST e inserí-las na fila de mensagens para serem consumidas pelos consumidores registrados.

Listing 1: Exemplo de regra

```
SELECT 6378137 * Math.sqrt(
    Math.pow((Math.toRadians(cast(Vehicle.lat,
    double)) - Math.toRadians(cast(
    BikePosition.lat, double))), 2) +
    Math.pow((Math.toRadians(cast(Vehicle.
    lng, double)) - Math.toRadians(cast(
    BikePosition.lng, double))), 2))
AS distance, bikePosition.imei,
    bikePosition.lat, bikePosition.lng,
    Vehicle.imei, Vehicle.lat, Vehicle.lng
FROM BikePosition.win:length(1)
AS bikePosition, Vehicle
ORDER BY distance ASC LIMIT 1
```

Foi desenvolvida uma aplicação Android para o case de acidentes de trânsito ocorridos com bicicletas, com intuito de consumir dados da plataforma e apresentar informações sensíveis ao contexto. O aplicativo possui duas funcionalidades básicas, providas através do consumo de dados do Módulo de *Web Services*. Um desses serviços é o "Alerta de *Crowdsensing*", onde o aplicativo envia sua posição de GPS atual e recebe a informação da distância entre o ciclista e os pontos de acidentes registrados pelo sistema de *Crowdsensing*, alertando ao ciclista que ele está próximo a um ponto de acidente. Outro serviço é o "Alerta de CEP", que em tempo real pode identificar a aproximação de algum usuário que esteja usando o aplicativo no modo veículo motorizado, e ser alertado a fim de evitar possíveis colisões.

4. AMBIENTE DE TESTE

Esta seção discute os materiais e métodos utilizados para a prova de conceito da plataforma proposta. Para a execução dos testes foram utilizados dois computadores, cuja configuração está descrita na tabela 1.

Tabela 1: Configuração das máquinas utilizadas

	Processador	Frequência	RAM
M.USER	2410M	2,3GHz	DDR3 8GB
M.SERVER	M380	2,53GHz	DDR3 4GB

O computador chamado M.USER foi utilizado para simular usuários e enviar status e requisições à plataforma. A simulação das requisições foi realizada em M.USER utilizando a ferramenta de testes de performance JMeter[1] versão 2.9. O computador chamado M.SERVER, foi aquele que hospedou a plataforma, recebeu e processou as requisições de M.USER.

Ao todo foram realizados dois testes. O primeiro teste (a partir daqui chamado de T1) simulou apenas requisições onde os usuários não passavam por pontos de acidentes, previamente cadastrados pelo serviço de *Crowdsensing*. No segundo teste (a partir daqui chamado de T2), em todas as requisições os usuários estavam em rota de perigo, segundo o serviço de *Crowdsensing*, e estavam em zona de colisão com outro usuário. A Seção 5 descreverá o resultado de cada teste.

Considerando a natureza em tempo real dos dados trafegados pela plataforma, dada a necessidade de manter atualizada a posição geográfica dos usuários, para a configuração dos usuários virtuais no JMeter foram definidos os seguintes parâmetros: *Threads* = `1000`, *Ramp-up* = `2` e *LoopCount* = `forever`. Ou seja, indefinidamente, a cada 2 segundos 1000 usuários virtuais enviam requisições à plataforma. Em cada teste o JMeter aferiu o desempenho da plataforma recebendo requisições simultâneas dos 1000 usuários virtuais durante 15 minutos. A performance foi monitorada pelas seguintes métricas: *Times Over Time, Transactions per Second, Response Times Percentiles, Time Graph.*

As requisições dos usuários virtuais foram do tipo GET e foram configuradas com os seguintes parâmetros: *Connect Timeout* = `10000` (ms), *Response Timeout* = `10000` (ms). O *Path* de conexão para envio das requisições segue a segue o seguinte formato: `/project/rest/trace/[id da fila]/[latitude]/[longitude]/`, como por exemplo: `/project/rest/trace/54321/-8.05529673777147/-34.951613545417786/`.

Além disso, em M.SERVER foi feita uma configuração para lidar com a grande quantidade de requisições dos usuários virtuais e diminuir o custo para a abertura de conexões. Através da biblioteca C3PO[2] do hibernate[3] foi configurado um *pool* de 1000 conexões.

5. RESULTADOS DOS TESTES

Nesta seção serão consolidados os resultados dos dois testes realizados.

Como apresentado anteriormente, no teste T1 foi simulado um total de 1000 usuários enviando seu status ao servidor e esperando alguma resposta sobre a situação da rota em curso. Neste teste, 100% dos usuários estavam em rota segura, ou seja, não haviam notificações do serviço de *Crowd-Sensing*. Portanto, nenhum evento foi disparado pelo serviço de CEP e nenhuma mensagem foi adicionada à fila de mensagens do usuário no RabbitMQ. Por outro lado, no teste T2 100% dos usuários estavam em uma rota que tinha alguma notificação pelo serviço de *CrowdSensing*. Dessa forma, a cada requisição, um evento CEP era disparado e uma mensagem era adicionada à fila de mensagens do usuário com a descrição do acidente logo à frente.

Dessa forma, o teste T1 possui uma característica exclu-

[1]Disponível em http://jmeter-plugins.org
[2]Disponível em http://www.mchange.com/projects/c3p0/
[3]Disponível em http://hibernate.org/

sivamente síncrona. Neste teste através de um *web service RESTful* provido pela plataforma o usuário envia sua coordenada geográfica e apenas espera um documento JSON com a distância e as características do ponto de acidente mais próximo. De fato, nenhum usuário neste teste estava em rota de risco. O Teste T2, por sua vez, possui características síncrona e assíncrona. Além da conexão síncrona do teste T1, o teste T2 calcula a distância do ciclista em relação ao automóvel mais próximo. Em caso de ambos estarem abaixo de uma distância tolerável o serviço de eventos disparava alertas nas filas de mensagens do ciclista e do automóvel. Dessa forma, ambos mantinham uma conexão assíncrona com a plataforma para serem notificados da proximidade de outro usuário.

5.1 Tempo de resposta

O objetivo desta métrica foi verificar o tempo médio que a resposta a uma requisição demora para chegar até o usuário virtual. Ou seja, o tempo que inicia quando a requisição é enviada ao servidor e que termina quando a resposta é completamente recebida pelo cliente. No JMeter é representada pela métrica *Times Over Time* e *Time Graph*.

Como esperado, em geral o tempo de resposta de T2 foi superior ao tempo de T1. Como ilustrado pela Figura 4, o tempo médio de resposta para T1 foi de 3708,58 ms e 4660,02 ms para T2. De fato, tomando 30 amostras aleatórias de tamanho 100, com reposição nos tempos de respostas mensurados, e considerando um nível de 95% de confiança, tem-se um intervalo de tempo médio de resposta de 3604.47ms a 3840.0ms para o teste T1 e de 4509.861ms a 4627.493ms para o teste T2.

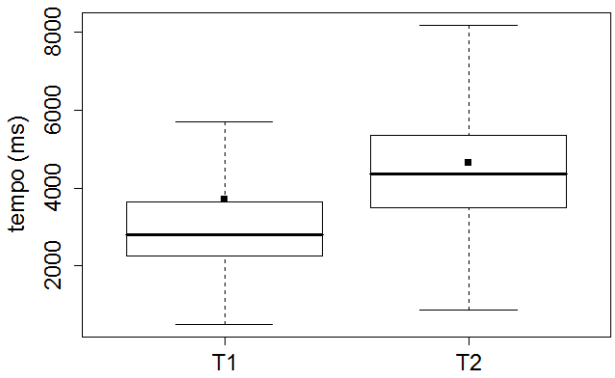

Figura 4: Distribuição dos tempos de resposta. Dada a variação apresentada nas medições de tempo, sua representação ficou melhor detalhada em um gráfico em caixa. A linha preta, como padrão, representa a mediana e o quadrado em preto representa a média.

Ao contrário de T1, em T2 há o tempo dedicado à execução do módulo de processamento de eventos complexos e de notificações nas conexões assíncronas com o RabbitMQ. Através das médias elencadas anteriormente, pode-se calcular que o tempo gasto com essas tarefas é de, em média, 951,44 ms. Ou seja, esse é o tempo médio que a plataforma leva para receber um fluxo de coordenadas geográficas, calcular a distância desta em relação aos pontos de acidentes, verificar se este ponto está em uma zona de risco e, por fim, disparar notificações nas filas de mensagens do RabbitMQ.

Para entender melhor o tempo de resposta, a Figura 5 mostra o percentil dos tempos de resposta de cada um dos testes. Pode-se observar que o tempo médio de resposta de T1 está no percentil 79,0%. Ou seja, 79,0% das requisições tem um tempo menor ou igual à média. Da mesma forma o tempo médio de resposta de T2 está no percentil 68,8% (ver Figura 5). No JMeter o percentil é dado pela métrica *Response Times Percentiles*.

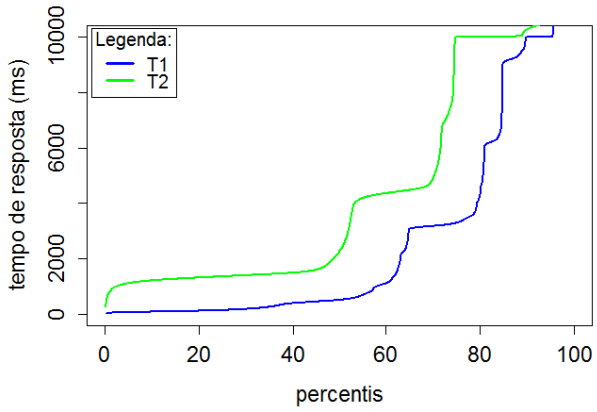

Figura 5: Percentil do tempo de resposta. Os valores de percentil ordenam crescentemente a amostra em 100 partes. Dessa forma, o n-ésimo percentil representa os primeiros n% dos valores da amostra.

5.2 Taxa de sucesso e falha

O objetivo desta métrica foi evidenciar o comportamento de sucesso e de falha nas requisições nos teste T1 e T2. Como mencionado na Seção 4, as requisições do JMeter foram configuradas com os parâmetros de *Connect Timeout* = `10000` (ms) e *Response Timeout* = `10000` (ms). Ou seja, caso uma conexão demorasse mais que dez segundos para ser aberta ou caso a requisição demorasse mais que dez segundos para ser respondida, esta requisição era considerada falha. No JMeter é representada pela métrica *Transactions per Second*.

Observando a Figura 6, pode-se notar que para o teste T1 houve uma média de 313,31 requisições por segundo processadas com sucesso e 21,02 requisições por segundo falhas. Já para o teste T2 houve uma média de 189,14 requisições por segundo concluídas com sucesso e 32,53 requisições por segundo falhas. De fato, assim como na Seção 5.1, tomando 30 amostras de tamanho 100 dos tempos de respostas mensurados e considerando um nível de confiança de 95%, tem-se um intervalo de médio de 306.78 a 317.32 requisições bem sucedidas por segundo no teste T1 e 186.93 191.29 para o teste T2. Já para as falhas, os intervalos de confiança são de 19.77 a 21.34 requisições falhas no teste T1 e 31.22 a 33.28 falhas para o teste T2.

No teste T2, para cada requisição respondida com sucesso, duas mensagens eram enviadas ao RabbitMQ, sendo uma para a fila de mensagens do ciclista e outra à fila de mensagens do automóvel. Acessando no servidor o endereço `localhost:15672/#/`, é possível encontrar o painel de controle do RabbitMQ e acompanhar em tempo real as atividades do serviço de fila de mensagem. Ao todo foram enviadas

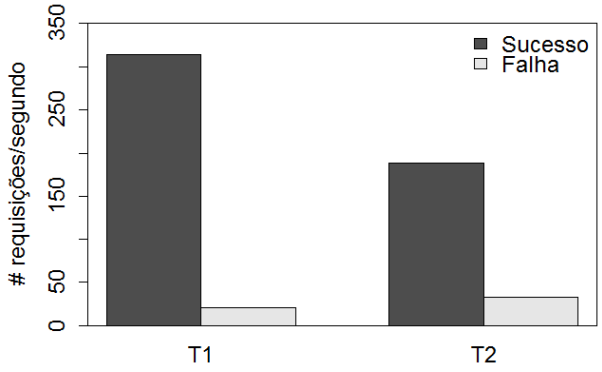

Figura 6: Taxa de sucesso e de falha nas requisições. As barras escuras representam as requisições concluídas com sucesso. As barras claras representam as requisições falhas.

350.307 mensagens ao RabbitMQ, correspondendo a uma média de 389,23 mensagens por segundo (ver Figura 7). De fato, esta taxa é praticamente o dobro da quantidade de requisições concluídas com sucesso em T2.

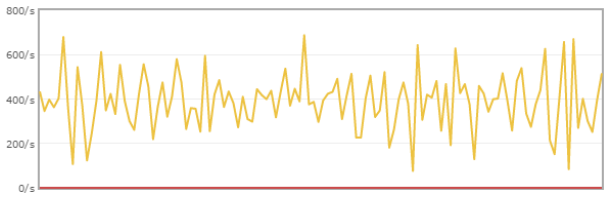

Figura 7: Taxa mensagens publicadas na fila do RabbitMQ por segundo. O período ilustrado corresponde aos últimos dez minutos de teste.

6. RESULTADOS E DISCUSSÃO

Como ilustrado na Seção 5, considerando um total de 1000 requisições simultâneas a plataforma apresentou um tempo médio de resposta de 3708,58 ms para rota segura (teste T1) e um tempo médio de 4660,02 ms para notificação de proximidade com outros usuários (teste T2). Sendo que, para T1, 79% das requisições tem um tempo menor ou igual à média e para T2, 68,9%. Além disso, de uma média de 334,33 requisições por segundo, para T1, apenas 6,28% delas não foram concluídas com sucesso. E para uma média de 221,67 requisições por segundo em T2, 14,67% foram requisições falhas.

Em um confronto dos resultados obtidos com uma demanda real de requisições, pode-se usar como exemplo o projeto BikePE[1]. Nesta iniciativa, 800 bicicletas estão disponíveis para aluguel em 80 estações espalhadas pela Região Metropolitana de Recife. A ideia é que através de uma opção sustentável se possa melhorar a mobilidade urbana e ao mesmo tempo oferecer mais opções de lazer à população.

Segundo BikePE[1], de segunda a sexta-feira há uma demanda média de 1100 viagens. Ou seja, apesar de diferentes contextos de aplicabilidade, e considerando os dados de desempenho apresentados pela plataforma pode-se perceber que utilizando apenas um computador com configurações domésticas seria o suficiente para suprir a demanda de um projeto desta dimensão em uma cidade como Recife utilizando a solução proposta neste trabalho.

Além disso, em caso de haver uma maior demanda de requisições, a plataforma pode ser instanciada em diferentes máquinas, oferecendo então a possibilidade de escala horizontal. Ou seja, para atender a uma demanda cada vez maior, não é necessário melhorar o hardware da máquina que hospeda a plataforma. Na verdade, instâncias da plataforma podem ser replicadas em várias máquinas com configurações tradicionais e assim obter uma maior escala a um menor custo.

7. CONCLUSÃO E TRABALHOS FUTUROS

Este artigo apresentou a proposta e a validação de uma plataforma que, ao fazer uso de componentes *off-the-shelves*, atende e implementa de forma eficaz e simplificada uma plataforma de middleware para solução de *Smart Cities*. Dentre suas características, estão: ser configurável para vários domínios, utilizar Processamento de Eventos Complexos para análise de *streams* de dados em tempo real, integrar sensores através de serviços Restful e de fila de mensagem e utilização de sensores humanos através do uso de *crowdsensing*.

A partir da análise dos resultados, pode-se concluir que a plataforma comportou-se de maneira satisfatória nos testes realizados, uma vez que suportou uma carga de requisições semelhantes às demandas reais, considerando que a máquina que hospedou a plataforma tinha a configuração típica de um computador doméstico.

Como ameaças à validação, tem-se o fato de não ter sido utilizada uma rede de dados 3G para a comunicação entre os usuários virtuais e a plataforma. No entanto, como os testes tiveram o objetivo de provar a possibilidade de desenvolvimento da plataforma, não foi necessário fazer testes de latência de rede, por exemplo. Dessa forma, preferiu-se optar pela simulação em uma rede *wireless* local.

De fato, testes de laboratório não reproduzem em sua totalidade o ambiente real de produção. Evidencia-se então a necessidade de realização de um estudo de caso com aplicação a acidentes de trânsito ocorridos com bicicletas. O objetivo é avaliar com maior precisão a eficiência da plataforma, analisando fatores como a latência de rede e pontos de saturação.

Por fim, o presente trabalho prova o conceito de que é possível construir uma plataforma de *middleware* para *smart cities*, baseada em localização, utilizando processamento de eventos complexos, parametrizáveis para vários domínios e capaz de processar regras elaboradas pelos próprios usuários.

Como trabalhos futuros pretende-se ampliar a integração de dados de fontes dos mais variados tipos de sensores e dispositivos, para vários tipos de protocolos de comunicação, além de ampliar a visualização de funcionalidades através do painel de dados estatísticos, apresentando dados em tempo real.

8. AGRADECIMENTOS

Este trabalho foi parcialmente financiado pelo Instituto Nacional de Ciência e Tecnologia para Engenharia de Software (INES[4]), CNPq e FACEPE[5] sob o processo nº IBPG-0773-1.03/13.

[4]http://www.ines.org.br
[5]http://www.facepe.br

9. REFERÊNCIAS

[1] Bikepe. http://www.bikepe.com/. Acessado em 1 Fev de 2015.

[2] Espertech. http://www.espertech.com. Acessado em 1 Jan de 2015.

[3] Rabbitmq, clustering guide. http://www.rabbitmq.com/clustering.html. Acessado em 1 Jan de 2015.

[4] Restlet - restful web framework for java. http://www.restlet.org/. Acessado em 1 Jan de 2015.

[5] Sencha touch - html5 mobile framework. http://www.sencha.com/products/touch/. Acessado em 1 Jan de 2015.

[6] United nations. world urbanization prospects: the 2011 revision. ny. Acessado em 1 Jan de 2015.

[7] E. Aubry, T. Silverston, A. Lahmadi, and O. Festor. Crowdout: A mobile crowdsourcing service for road safety in digital cities. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 86–91. IEEE, 2014.

[8] R. Borja and K. Gama. Middleware para cidades inteligentes baseado em um barramento de serviços. In *X Simpósio Brasileiro de Sistemas de Informação (SBSI)*, volume 1, pages 584–590, 2014.

[9] L. Brownsword, T. Oberndorf, and C. A. Sledge. Developing new processes for cots-based systems. *IEEE Software*, 17(4):48–55, 2000.

[10] M. Choi and B.-T. Kang. Html5 platform independent mobile application using location-based service. *Life Science Journal*, 11(7), 2014.

[11] T. Erickson. Geocentric crowdsourcing and smarter cities: Enabling urban intelligence in cities and regions. In *1st Ubiquitous Crowdsourcing Workshop at UbiComp*, 2010.

[12] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2010.

[13] K. Farkas, A. Z. Nagy, T. Tomás, and R. Szabo. Participatory sensing based real-time public transport information service. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 141–144. IEEE, 2014.

[14] L. Filipponi, A. Vitaletti, G. Landi, V. Memeo, G. Laura, and P. Pucci. Smart city: An event driven architecture for monitoring public spaces with heterogeneous sensors. In *Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on*, pages 281–286. IEEE, 2010.

[15] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39, 2011.

[16] F. Gasperoni. Floss, cots, and safety: A business perspective. *3rd European Congress ERTS – Embedded Real Time Software*, 2006.

[17] J.-L. Maréchaux. Combining service-oriented architecture and event-driven architecture using an enterprise service bus. *IBM Developer Works*, pages 1269–1275, 2006.

[18] T. Nagel, M. Maitan, E. Duval, A. V. Moere, J. Klerkx, K. Kloeckl, and C. Ratti. Touching transport-a case study on visualizing metropolitan public transit on interactive tabletops. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, pages 281–288. ACM, 2014.

[19] M. Naphade, G. Banavar, C. Harrison, J. Paraszczak, and R. Morris. Smarter cities and their innovation challenges. *Computer*, 44(6):32–39, 2011.

[20] N. Odendaal. Information and communication technology and local governance: understanding the difference between cities in developed and emerging economies. *Computers, Environment and Urban Systems*, 27(6):585–607, 2003.

[21] W. Sherchan, P. P. Jayaraman, S. Krishnaswamy, A. Zaslavsky, S. Loke, and A. Sinha. Using on-the-move mining for mobile crowdsensing. In *Mobile Data Management (MDM), 2012 IEEE 13th International Conference on*, pages 115–124. IEEE, 2012.

[22] D. A. Wheeler. Why open source softwarefree software (ossfs, floss, or foss)? look at the numbers! http://www.dwheeler.com/oss_fs_why.html. Acessado em 1 Jan de 2015.