Passages de Paris 10 (2015) 301-311

www.apebfr.org/passagesdeparis

SENSORIAMENTO PARTICIPATIVO PARA DETERMINAÇÃO DE PRIORIDADES NO ATENDIMENTO DE VÍTIMAS DE DESASTRES NATURAIS

André Leon Sampaio GRADVOHL*

Resumo: *Smartphones* têm sido usados para várias atividades, desde o envio de mensagens curtas até participação nas redes sociais, inclusive em microblogs, como o Twitter. A participação em redes sociais para relatar eventos do cotidiano criou uma nova linguagem baseada nos chamados *hashtags*. A sintaxe de um *hashtag* é formada pelo símbolo # seguido por uma palavra que adiciona uma semântica à frase que o antecede. Neste trabalho, utilizamos o conceito de sensoriamento participativo para captar informações sobre pessoas em situação de emergência após a ocorrência de desastres naturais. Trata-se de um sistema para processamento online de fluxos de dados que observa as mensagens enviadas pelo microblog e, com base nas *hashtags* enviadas, determina os locais com maior prioridade no atendimento das vítimas.

Palavras-chave: Sistemas distribuídos; processamento online de fluxos de dados; tolerância a falhas; situações de desastres naturais; atendimento a vítimas.

Abstract: Smartphones are suitable for many activities, from sending short messages through share personal information on social networks, including on microblogs like Twitter. Participation in social networks to report events of everyday life created a new language based on so-called hashtags. The # sign followed by a word that adds semantics to the sentence that precedes it form the syntax of a hashtag. In this work, we use the concept of participatory sensing to capture information about people in emergency after the occurrence of a natural disaster. It is a system for online data stream processing, which observe the messages sent to the microblog and, based on hashtags within, determines the locations with the highest priority victims to care.

Keywords: Distributed systems; online data stream processing; fault tolerance; natural disaster situations; victims care.

I. INTRODUÇÃO

Processamento de fluxo de dados ou *event stream processing* (ESP) é uma iniciativa que vem crescendo nos últimos anos com as redes sociais e outras aplicações que requerem processamento de dados em tempo real. Exemplos de tais aplicações são a detecção de ataques a redes de computadores, análises financeiras, filtragem de *spam*, publicidade direcionada, análise de tendências, sensoriamento participativo e até mesmo

*André Leon Sampaio GRADVOHL é doutor em Engenharia Elétrica e Computação pela Universidade Estadual de Campinas (UNICAMP) e especialista em Jornalismo Científico, também pela UNICAMP. Atualmente realiza estudos de Pós-doutorado no *Laboratoire d'Informatique* de Paris 6, na *Université Pierre et Marie Curie* – Paris VI. E-mail: gradvohl@ft.unicamp.br.

Texto recebido em 05 de junho de 2014.
Aceito para publicação em 25 de junho de 2014.

situações de gestão de desastres, entre outros (Adam, Eledath e Mehrotra 186), (Adam, Shafiq e Staffin, "Spatial Computing and Social Media in the Context of Disaster Management" 91), (Boulos, Resch e Crowley 1) e (Burke, Estrin e Hansen 117). A principal característica de um aplicativo que usa processamento de fluxo de eventos é lidar com um fluxo contínuo de dados de entrada, o mais rápido possível, reduzindo este grande volume de dados de entrada, para a tomada de decisão ou de armazenamento do que é relevante.

Por outro lado, o conceito sensoriamento participativo se refere à formação de grupos de pessoas que contribuem com informações sensoriais sobre o ambiente em que estão, por exemplo, informações sobre calor, frio, ventos e estado do tráfego de veículos, entre outras informações. A participação desses grupos em redes sociais para relatar eventos do cotidiano criou uma nova linguagem baseada nos chamados *hashtags*. A sintaxe do *hashtag* é formada pelo símbolo # seguido por uma palavra que pretensiosamente adiciona uma semântica à frase que antecede o próprio *hashtag*.

Na ocorrência de desastres naturais, e. g. terremotos, incêndios e inundações, as pessoas atingidas por esses desastres poderiam enviar mensagens nos microblogs informando sua situação física e localização através de *hastags* específicos. Ações desse tipo ocorreram em algumas crises recentes (Potts, Seitzinger e Jones 235). Nessa situação, propomos o desenvolvimento de um software que obtenha essas mensagens, filtre-as de acordo com as *hashtags* utilizadas e consiga indicar as prioridades nos atendimentos dessas pessoas.

Esse software deve ser tolerante a falhas estruturais, isto é, componentes que deixam de funcionar, e falhas Bizantinas[1], ou seja, falhas na determinação das prioridades causadas por excesso de mensagens de usuários com melhores condições para o envio de informações.

II. SISTEMAS ESP

Para desenvolver esse software utilizamos como base um sistema ESP distribuído. Em poucas palavras, um sistema ESP recebe um fluxo de dados como entrada, realiza alguns cálculos ou transformações e produz uma saída.

Em um sistema ESP, um fluxo contínuo de dados passa por uma topologia (grafo de fluxo) para ser processado e produzir resultados. Essa topologia é um grafo acíclico dirigido, onde as arestas (setas) são conexões de dados e os vértices (círculos) são operadores, conforme ilustrado na Figura 1. Os operadores realizam cálculos ou

[1] O adjetivo "Bizantina" vem do chamado "Problema dos Generais Bizantinos", uma metáfora utilizada na Ciência da Computação para ilustrar um problema de sistemas distribuídos. Nesse problema, informações conflitantes vindas de diferentes partes de um sistema podem induzir uma falha no sistema inteiro.

transformações nos fluxos de entrada para produzir novos fluxos para outros operadores ou para saída (Hirzel, Andrade e Gedik 5).

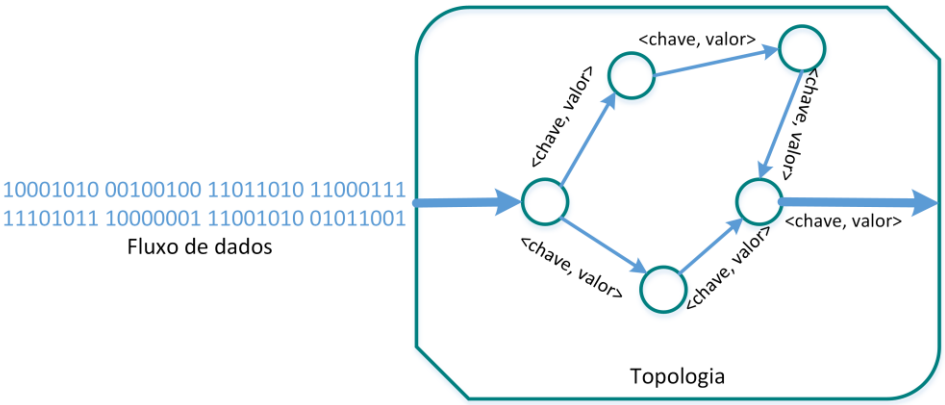

Figura 1: Topologia de um sistema ESP distribuído.

O fluxo de dados de entrada é dividido em itens de dados ou eventos. Esses itens ou eventos consistem em um par chave-valor que forma uma tupla[2]. A chave identifica a tupla e o valor é uma sequência de *bytes* associados a uma chave particular. Observe que alguns operadores precisam manter seu estado. Isso significa que o operador deve manter a informação já coletada enquanto processa diferentes tuplas que estão chegando.

Assim que os operadores recebem as tuplas e pouco antes de seu processamento, as tuplas são armazenadas temporariamente nas filas de entrada (uma para cada fluxo de entrada) em cada operador. Da mesma forma, as filas de saída mantêm as tuplas geradas por cada operador, pouco antes de outros operadores recebê-las.

A quantidade de dados de entrada por unidade de tempo não está sob o controle do sistema, que geralmente recebe esse fluxo em alta velocidade. A quantidade de dados que vem do fluxo de entrada é geralmente tão grande que não é prático armazenar esse fluxo em discos para processamento posterior.

Os eventos ou fluxo de dados entrada podem vir de diferentes fontes, como formulários na *world wide web*, telefones celulares ou sensores específicos. Manter este sistema em um único servidor é arriscado e este pode não ter o poder de computação suficiente para atender a demanda do sistema (Martin, Fetzer e Brito 23). Portanto, um sistema distribuído para este fim aumentará a capacidade de processamento e a disponibilidade do sistema como um todo.

[2] O termo "tupla" é muito comum na Ciência da Computação e refere-se a uma lista de elementos associados. Por exemplo, a associação entre o número de identidade de uma pessoa e seu nome forma uma tupla.

Por outro lado, as aplicações de processamento de fluxo geralmente são executadas por um longo período de tempo e devem manter seu estado, isto é, as informações temporárias calculadas até então, durante períodos prolongados. Assim, ao executar este tipo de aplicação, já devemos esperar que o sistema enfrentará problemas, como falhas, atualizações de infraestrutura, manutenções agendadas e atualizações.

De acordo com Hummer, Inzinger e Leitner (172), estouro de *buffer* e falhas de unidades de processamento (nós) são as duas falhas mais desafiadoras em sistemas ESP distribuídos. Um estouro de *buffer* ocorre quando uma unidade de processamento não pode alocar memória suficiente para tratar o evento de entrada. Por sua vez, uma falha de nó ocorre quando o hardware que suporta um nó de processamento apresenta falhas.

Portanto, para atingir os seus objetivos, é necessário construir um sistema distribuído, tolerante a falhas, que mantenha estados persistentes e escalável. A característica de ser tolerante a falhas é importante para manter o sistema trabalhando tanto tempo quanto possível, uma vez que a perda de uma parte do fluxo de dados pode levar a decisões incorretas. Manter estados persistentes do sistema também é necessário para reiniciar o sistema rapidamente após uma situação de falha. Além do mais, a escalabilidade é uma característica que permite que o crescimento do sistema, a adição ou remoção de componentes de processamento não causem impactos no seu desempenho.

Os primeiros sistemas ESP, criados no início dos anos 2000, foram o Aurora, Boreal, STREAM, TelegraphCQ, NiagaraCQ e Cougar (Gulisano 155). Naquela época, os sistemas eram centralizados, ou seja, eles eram executados em um único servidor, e seu objetivo era superar os problemas para lidar com processamento online de fluxo de dados dos bancos de dados tradicionais.

É importante mencionar que os objetivos dos bancos de dados são essencialmente diferentes dos ESPs. Enquanto os bancos de dados são otimizados para o armazenamento e consulta eficientes em tabelas de dados, os ESPs são projetados para fornecer uma análise de fluxos de baixa latência (pequena demora no processamento de dados) e alto desempenho. Assim, os requisitos de tolerância a falhas e escalabilidade diferem substancialmente em ambos os tipos de sistemas.

III. A TOLERÂNCIA A FALHAS

A maneira mais ou menos suave com que um sistema de computador degrada seu desempenho na presença de falhas de alguns de seus componentes é o que chamamos de modelo de tolerância a falhas. Assim, as técnicas para tolerância a falhas usadas dependerão do modelo de tolerância previsto para sistemas de computação específicos. De acordo com Treaster (2), existem dois principais modelos de falhas, o modelo de falhas bizantinas e o modelo de falhas parada total. Mas há também um terceiro modelo chamado falhas intermitentes.

No modelo de falha bizantina, os nós que apresentam falhas ainda estão ativos e interagem com o restante do sistema. No entanto, suas mensagens são corrompidas, embora ainda sejam válidas. Isto significa que os outros nós do sistema não conseguem detectar as falhas, nem os nós falhos (Costa, Pasin e Bessani, "Byzantine Fault-Tolerant MapReduce: Faults are Not Just Crashes" 33) (Costa, Pasin e Bessani, "On the Performance of Byzantine Fault-Tolerant MapReduce" 307).

Por outro lado, na falha parada total, quando as falhas acontecem, um nó deixa de produzir saídas e interagir com o sistema. Este é um tipo de falha fácil de detectar e superar. O terceiro modelo, intermitente, é baseado em uma extensão do modelo de falhas parada total. Neste caso, este modelo inclui também falhas de desempenho, que ocorrem quando um nó do sistema começa a apresentar desempenho muito pior do que os demais.

Para melhorar a tolerância às falhas estruturais, três técnicas podem ser utilizadas:

- Verificações (*checkpoints*) periódicos, coordenados ou não coordenados: referem-se ao armazenamento do estado de cada um dos nós da topologia (Goswami e Sahu 138).
- *Upstream backup*: consiste na retenção dos eventos na fila de saída de cada um dos operadores da topologia até que o operador que receberá esses eventos reconheça o processamento dos eventos na sua fila de entrada (Fernandez, Migliavacca e Kalyvianaki, "Integrating Scale out and Fault Tolerance in Stream Processing Using Operator State Management" 733).
- Replicação de operadores: relaciona-se à manutenção de cópias dos operadores de forma que, na ocorrência de uma falha em um operador, o segundo operador possa manter o sistema em funcionamento (Fernandez, Migliavacca e Kalyvianaki, "Scalable and Fault-tolerant Stateful Stream Processing" 15).

Alguns sistemas utilizam essas técnicas isoladamente ou combinadas. No entanto, é preciso observar que o uso dessas técnicas aumenta a latência do sistema e degrada seu desempenho. Portanto, é uma solução que exige análises comparativas entre desempenho e robustez do sistema antes de ser considerada.

IV. A ARQUITETURA DO SISTEMA PROPOSTO

O sistema para sensoriamento participativo para determinação de prioridades no atendimento de vítimas de desastres naturais que propomos é composto por dois softwares diferentes. Um dos softwares, designado software cliente, será embarcado no *smartphone* do usuário. O outro software, chamado oráculo, implementará o ESP e estará em um *cluster* computacional, isto é, em um computador composto por várias unidades de processamento interligadas por uma rede de alta velocidade. É preciso que o oráculo seja um software distribuído e tolerante a falhas.

O software cliente poderá vir pré-instalado, isto é, instalado na própria fábrica do *smartphone*, e ser ativado pelo próprio usuário quando estiver em uma situação de desastre natural. Esse software funcionará como um comunicador instantâneo e permitirá que o usuário utilize *hashtags* específicas para descrever como está sua situação física, sua situação psicológica, as necessidades de atendimento médico, de bombeiros, ou de suporte policial.

Junto com as mensagens que o usuário eventualmente enviará, estão também outros dados relativos à sua identidade e à sua localização. Esses dados podem ser, por exemplo, número de telefone, coordenadas de *Global Positioning Systems* (GPS) e altitude em relação ao solo, entre outros. Essas informações geralmente são captadas pelo próprio *smartphone* que possui sensores específicos para captura-las, conforme ilustra a Figura 2. O software cliente ainda receberá informações do oráculo, informando se o usuário será atendido e o tempo mínimo para o atendimento.

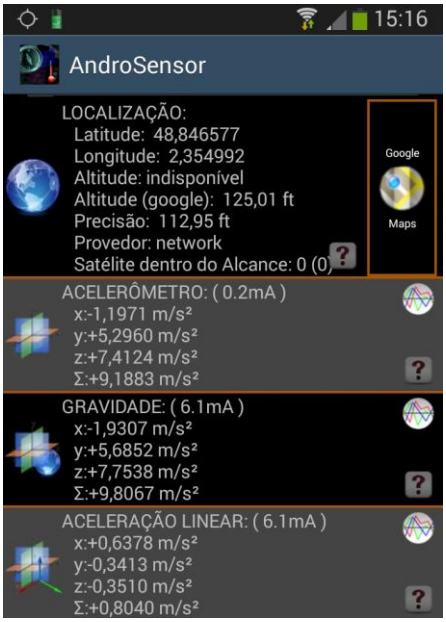

Figura 2: Exemplos de sensores disponíveis em um smartphone.
Fonte: Software AndroSensor[3].

Dependendo do alcance do desastre natural, a quantidade de mensagens enviadas pelos usuários do software cliente será enorme. A expectativa é que se tenha cerca de 5,5 mil mensagens por segundo. Esse foi o número médio de mensagens no Twitter quando aconteceram o tsunami em março de 2011 no Japão e o terremoto da costa oeste dos Estados Unidos da América em agosto de 2011. Daí a necessidade de se ter um software tolerante a falhas para suportar esse número de mensagens.

[3] O software AndroSensor foi desenvolvido para dispositivos móveis com o sistema operacional Android e está disponível no site http://www.fivasim.com/androsensor.html.

Outra questão que deve ser observada é a qualidade da conexão à internet logo após a ocorrência do desastre natural. Geralmente as conexões à rede através de cabos são comprometidas, pois com o desastre os cabos se rompem. Nesse caso, pontos de acesso com tecnologia de rede sem fio para acesso à internet podem ser distribuídos nas áreas do desastre. A distribuição pode ser via aérea, com lançamento a partir de helicópteros, ou pelas próprias equipes de emergência, ao transitarem pela área do desastre.

Em relação ao sistema proposto, na ocorrência de um desastre natural, o oráculo se conectará ao microblog Twitter e passará a coletar todas *hashtags* relacionadas ao desastre natural que está em observação. Uma topologia será implementada no oráculo para filtrar as mensagens do microblog e organizá-las a partir dos seguintes dados:

- Identificador do usuário, por exemplo, número do telefone.
- Localização do usuário (latitude, longitude e altitude).
- Situação física (pode se locomover ou não, pode guiar outras vítimas, possui ferimentos leves, possui ferimentos graves).
- Situação psicológica (calmo, agitado, muito nervoso).
- Necessidades de atendimento (médicos, bombeiros, policiais).

Com base nesses dados, algoritmos implementados no oráculo determinarão a prioridade no atendimento das vítimas e o tempo médio para atendimento, com base na localização das equipes de emergência.

No cenário descrito na Figura 3, os círculos maiores são os *hotspots* que são *smartphones* com melhor qualidade de transmissão de dados e maior carga de bateria. Essa qualidade de transmissão é determinada pelo próprio aparelho ao captar o sinal que o conecta à internet. Os círculos menores também são *smartphones*, porém com baixa qualidade de recepção do sinal da internet diretamente, mas com boa conexão com um *hotspot*.

As setas na Figura 3 ilustram as trocas de mensagens entre os *smartphones* e o oráculo através do Twitter, que não está representado nessa figura para manter simplicidade. Assim, ao presenciar a ocorrência de um evento (incêndio, desmoronamento, inundação), o usuário envia uma mensagem pelo Twitter através do software cliente embarcado no seu *smartphone*, relatando um incidente, suas condições físicas e psicológicas. Esse tipo de mensagem é indicado pelo ícone com o número 1 embutido no círculo vermelho.

As mensagens dos vários *smartphones* chegam aos *hotspots* que, por sua vez, podem agregar as mensagens antes de enviá-las ao oráculo para poupar energia. As mensagens agregadas são representadas pelo ícone com o número 2 embutido no círculo verde na Figura 3.

Ao receber as mensagens, o oráculo determinará os casos mais urgentes e designará as equipes de emergência com melhores condições, ou seja, aquelas que estão mais

próximas ao incidente, com maior velocidade de locomoção, com maior número de profissionais aptos para atender à ocorrência. Tomada essa decisão, o oráculo enviará diretamente aos *hotspots* uma mensagem com informações sobre o atendimento de cada usuário que enviou mensagens. A mensagem enviada pelo oráculo está representada pelo ícone com o número 3 embutido no círculo verde na Figura 3.

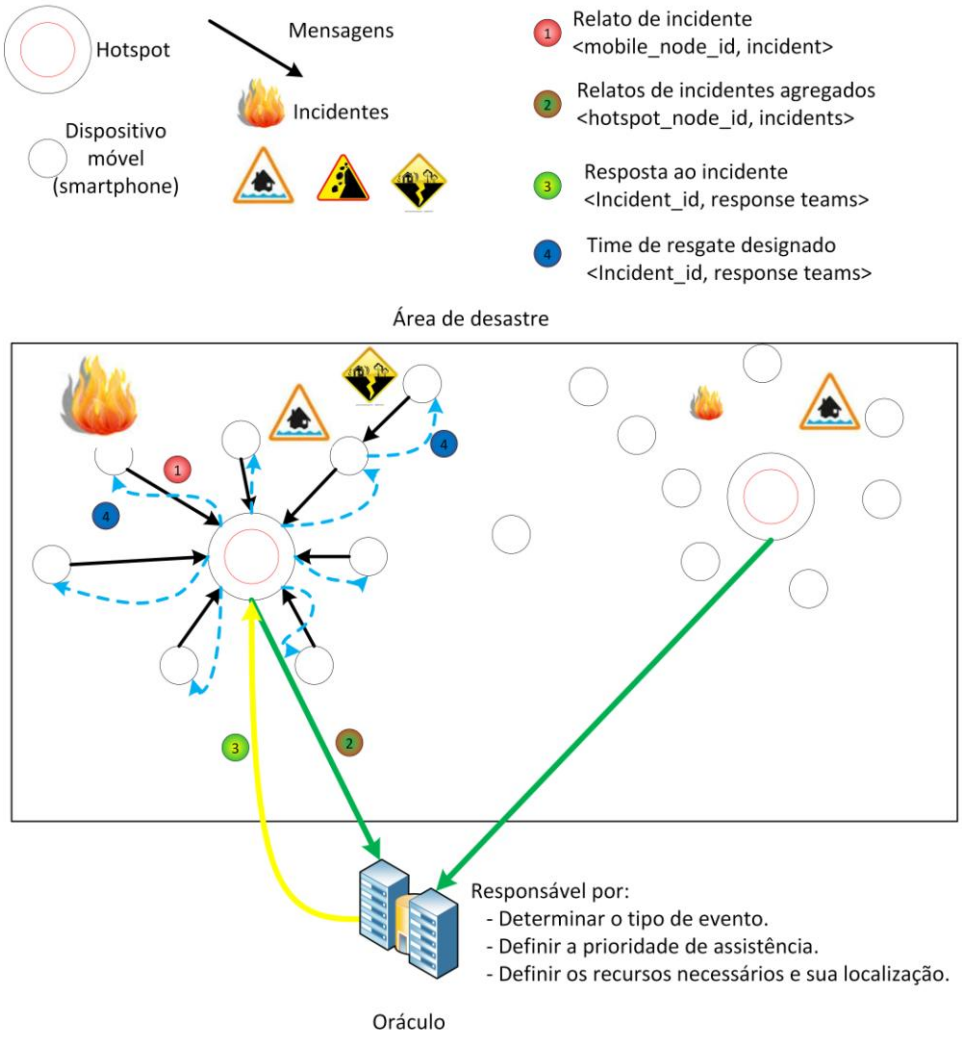

Figura 3: Ilustração do funcionamento do sistema.

Por fim, os *hotspots* encaminham as mensagens aos respectivos smartphones mais próximos com as informações providas pelo oráculo. A mensagem enviada pelo *hotspot* a cada um dos smartphones está representada pelo ícone com o número 4 embutido no círculo azul na Figura 3.

É preciso destacar que o sistema proposto é robusto e tolerante a falhas. O sistema é capaz de determinar quais os dispositivos móveis com melhor qualidade de sinal e

maior carga de bateria. Esses são eleitos como *hotspots*. No caso de falhas nesses *hotspots*, outros podem ser eleitos imediatamente.

Além disso, o sistema considera o consumo de energia dos dispositivos móveis, agregando mensagens nos próprios *hotspots*, antes do envio para o microblog ou ao receber mensagens do oráculo. Dessa forma, o número de mensagens é menor e a chance das mensagens chegarem aos seus destinatários é maior.

V. OUTROS SISTEMAS SIMILARES

Há outros sistemas similares, cujo intuito é ajudar a vítima de desastres naturais. Um deles é o Life 360[4] que se trata de um aplicativo que permite o compartilhamento da localização entre os entes de uma família. Neste caso, as informações são referentes apenas à localização das pessoas e não há um contato com as equipes de emergência.

Outro aplicativo é o Plerts[5] que também permite o contato com a família, enviando informações de GPS e áudio. No entanto, não há um contato direto com as equipes de emergência.

Ambos os aplicativos precisam de um servidor central para gerenciar e armazenar as mensagens e não utilizam um microblog de acesso global. Essa característica torna esses sistemas de uso exclusivo e, dessa maneira, as informações não são públicas, dificultando o uso dos sistemas pelos serviços estatais de emergência. Além disso, até onde se sabe, esses sistemas não consideram o consumo de energia nos dispositivos móveis.

VI. CONCLUSÃO

Ao propor um sistema que possa captar, analisar e classificar as mensagens de emergência em um microblog, estamos criando uma ferramenta que pode aproveitar os recursos disponíveis em *smartphones* para melhorar o atendimento às vítimas de desastres naturais. Aliás, o número de *smartphones* tem crescido nos últimos anos e isso faz com que esses dispositivos sejam usados por boa parte da população.

Em qualquer lugar, ferramentas desse tipo são muito úteis para incrementar o gerenciamento de crises provocadas por desastres naturais ou mesmo aqueles provocados por grande adensamento populacional. Futuras melhorias no sistema podem adicionar, inclusive, conselhos relativos às rotas de fuga que os usuários poderão utilizar, caso haja uma ocorrência.

[4] Disponível em https://www.life360.com
[5] Disponível em https://plerts.com

Nossa expectativa é que um protótipo desse sistema seja implementado até o final deste ano para que possam ser realizadas simulações e testes para verificar a viabilidade do sistema.

AGRADECIMENTOS

O autor agradece à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo investimento na sua formação pós-doutoral, através do projeto 11785/13-6; à Faculdade de Tecnologia da Universidade Estadual de Campinas, pelo apoio ao permitir o afastamento para realização de pesquisas na França; e ao *Laboratoire d'Informatique* de Paris 6, pelo apoio às pesquisas.

REFERÊNCIAS BIBLIOGRÁFICAS

Adam, Nabil R., Basit Shafiq e Robin Staffin. "Spatial Computing and Social Media in the Context of Disaster Management." IEEE Intelligent Systems 27.6 (2012): 90-96.

Adam, Nabil R., et al. "Social media alert and response to threats to citizens (SMART-C)." 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaborate Com). 2012. 181-89.

Akidau, Tyler, et al. "MillWheel: Fault-Tolerant Stream Processing at Internet Scale." Very Large Data Bases. 2013. 734-46.

Boulos, Maged Kamel, et al. "Crowdsourcing, citizen sensing and sensor web technologies for public and environmental health surveillance and crisis management: trends, OGC standards and application examples." International Journal of Health Geographics 10.1 (2011): 1-29.

Burke, Jeff, et al. "Participatory sensing." Workshop on World-Sensor-Web (WSW'06): Mobile Device Centric Sensor Networks and Applications. 2006. 117-34.

Costa, Pedro, et al. "Byzantine Fault-Tolerant MapReduce: Faults are Not Just Crashes." IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom). 2011. 32-39.

—. "On the Performance of Byzantine Fault-Tolerant MapReduce." IEEE Transactions on Dependable and Secure Computing 10.5 (2013): 301-13.

Fernandez, Raul Castro, et al. "Integrating Scale out and Fault Tolerance in Stream Processing Using Operator State Management." Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York, NY, USA: ACM, 2013. 725-36.

—. "Scalable and Fault-tolerant Stateful Stream Processing." 2013 Imperial College Computing Student Workshop. Ed. Andrew V. Jones e Nicholas Ng. Dagstuhl, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013. 11-18.

Goswami, Diganta e S. Sahu. "An Efficient Protocol for Checkpoint-Based Failure Recovery in Distributed Systems." Distributed Computing and Internet

Technology. Ed. R.K. Ghosh e Hrushikesha Mohanty. Vol. 3347. Springer Berlin Heidelberg, 2005. 135-44.

Gulisano, Vincenzo Massimiliano. "StreamCloud: An Elastic Parallel-Distributed Stream Processing Engine". Tese de Doutorado. Facultad de Informática, Universidad Politécnica de Madrid, Madrid, 2012.

Hirzel, Martin, et al. "IBM Streams Processing Language: Analyzing Big Data in motion." IBM Journal of Research and Development 57.3/4 (2013): 7:1-7:11.

Hummer, Waldemar, et al. "Deriving a Unified Fault Taxonomy for Event-based Systems." Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. New York, NY, USA: ACM, 2012. 167-78.

Martin, André, Christof Fetzer e Andrey Brito. "Active Replication at (Almost) No Cost." 30th IEEE Symposium on Reliable Distributed Systems (SRDS). 2011. 21-30.

Potts, Liza, et al. "Tweeting Disaster: Hashtag Constructions and Collisions." ACM International Conference on Design of Communication. Pisa: ACM, 2011. 235-40.

Treaster, Michael. "A Survey of Fault-Tolerance and Fault-Recovery Techniques in Parallel Systems." ACM Computing Research Repository (CoRR) 501002 (2005): 1-11. 23 de maio de 2014 <http://arxiv.org/abs/cs/0501002>.