# Processamento Distribuído de Eventos Complexos Aplicado à Detecção de Eventos no Trânsito em Tempo Real*

**Fernando Freire Scattone**[1], **Kelly Rosa Braghetto**[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)

{ffs,kellyrb}@ime.usp.br

***Resumo.*** *Uma técnica comumente usada para processar dados em tempo real é o Processamento de Eventos Complexos (CEP* - Complex Event Processing*). CEP permite a detecção de eventos de interesse a partir da identificação de padrões nos dados processados. Este trabalho apresenta uma implementação de arquitetura de microsserviços auto-escalável para CEP, apropriada para lidar com grandes fluxos de dados, como os gerados por sensores em cidades inteligentes. Um sistema que detecta problemas no tráfego das linhas de transporte público de ônibus de São Paulo foi projetado para testar a arquitetura.*

## 1. Introdução

Uma das técnicas mais adequadas para processar dados em tempo real é o Processamento de Eventos Complexos (CEP – *Complex Event Processing*). Ferramentas de CEP fornecem modos de declarar tipos de eventos detectados com a ocorrência de um padrão de chegada de dados. CEP considera que seus dados de entrada são eventos e usa operadores sobre estes eventos, como filtros e agregações, para disparar a detecção de outros eventos [Etzion and Niblett 2010]. CEP pode ser usado, por exemplo, para modelar a detecção de situações de interesse no mundo real a partir de medições específicas de sensores.

Atualmente, existem três principais soluções de CEP de código aberto: ESPER, Drools Fusion e Siddhi[1]. Elas são bibliotecas para a definição de tipos de eventos e ações para execução depois da detecção. Para executá-las em um ambiente distribuído, a abordagem mais usada é acoplá-las a uma ferramenta de processamento de *streams*, como o Apache Storm[2]. Nesse tipo de estratégia, a distribuição de processamento de eventos é definida como um grafo acíclico dirigido, no qual cada nó é um ambiente para processamento de eventos e cada aresta é um caminho para o fluxo de eventos entre ambientes de processamento distintos. Entretanto, não é permitida a criação ou destruição de nós em tempo de execução, limitando as possibilidades de balanceamento de carga.

Este trabalho apresenta uma implementação de arquitetura de microsserviços para distribuir a carga de processamento de eventos dinamicamente, de forma que o sistema auto-escale conforme o uso de recursos nos seus nós varie. Um experimento foi projetado para testar a latência e vazão desta solução, utilizando um cenário de monitoramento de tráfego nas linhas do sistema de transporte público de ônibus de São Paulo, SP.

*O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Ele também é parte do INCT da Internet do Futuro para Cidades Inteligentes financiado pelo CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001, FAPESP proc. 14/50937-1 e FAPESP proc. 15/24485-9.

[1]espertech.com/, drools.org/, github.com/wso2/siddhi/

[2]storm.apache.org/

## 2. Processamento de Eventos Complexos

Um sistema em CEP é composto por três tipos de entidades principais: *Produtores de Eventos*, responsáveis por enviar eventos para o sistema, *Consumidores de Eventos*, responsáveis por coletar eventos do sistema, e *Processadores de Eventos*, ou EPAs (*Event Processing Agents*), responsáveis pela detecção de ocorrência de eventos de um tipo específico [Luckham 2001]. A concatenação de vários EPAs forma uma *Rede de Processamento de Eventos* ou EPN (*Event Processing Network*).

Oito tipos de operadores podem ser usados para definir os tipos de eventos nos EPAs: Filtragem, Projeção, Translação, Divisão, Enriquecimento, Agregação, Composição e Detecção de Padrões [Etzion and Niblett 2010]. Alguns destes operadores não guardam estado durante o processamento e geram um evento de saída a cada nova entrada. No entanto, outros operadores precisam juntar dados de vários eventos de entrada do mesmo tipo, ou até de tipos distintos, para detectarem um evento de saída.

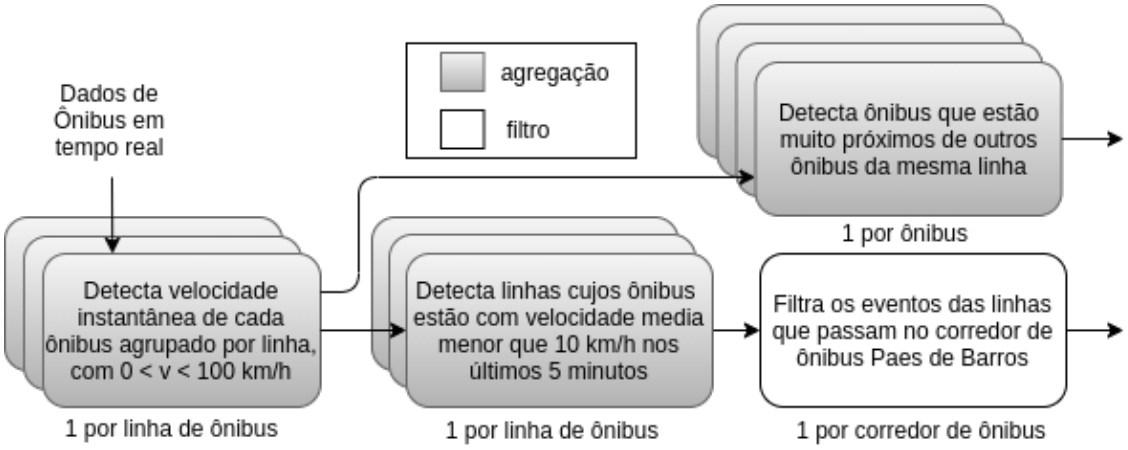


**Figura 1. Rede de processamento de eventos de localização de ônibus**

Um exemplo de EPN pode ser visto na Figura 1, onde os retângulos representam EPAs e as setas representam fluxos de eventos. O primeiro conjunto de EPAs da rede recebe em tempo real da SPTrans[3] dados de localização dos ônibus e, a partir deles, detecta a velocidade instantânea de cada ônibus, enviando as velocidades de todos os ônibus de mesma linha por um mesmo fluxo de eventos. Eles só geram eventos de saída caso a velocidade detectada esteja entre zero e 100 km/h. Essa "limpeza" precisa ser feita pois os dados recebidos da SPTrans podem conter erros e resultar velocidades incoerentes. O resultado desse processamento é usado na detecção de outros dois tipos de eventos.

Cada EPA da parte superior da Figura 1 detecta se ônibus de uma mesma linha estão circulando muito próximos uns dos outros, situação chamada de *bus-bunching*. Neste caso, cada EPA monitora um único ônibus e só dispara um evento caso outros ônibus da mesma linha estejam próximos ao ônibus monitorado. Outro tipo de situação é detectado usando os dois conjuntos de EPAs descritos na parte inferior da figura, à direita. No primeiro deles, tem-se novamente um EPA por linha de ônibus, para detectar a linhas cujos ônibus estão com uma velocidade média muito baixa nos últimos cinco minutos. Por fim, um outro EPA seleciona, dentre os eventos gerados na detecção das linhas com velocidade baixa, os dos ônibus que passam no corredor "Paes de Barros".

[3] A São Paulo Transporte S/A (SPTrans) faz a gestão do transporte público por ônibus em São Paulo-SP.

## 3. Arquitetura de Microsserviços para CEP Distribuído

Em um trabalho anterior [Freire Scattone and Rosa Braghetto 2018], propusemos uma arquitetura de microsserviços auto-escalável para a distribuição do processamento de eventos, em que cada instância de um microsserviço de CEP, denominado *cep-worker*, pode processar vários EPAs e o fluxo de eventos entre EPAs distintos é feito por um sistema de mensageria em outro microsserviço.

Neste trabalho, apresentamos uma implementação dessa arquitetura, que está sendo desenvolvida como uma extensão da plataforma de cidades inteligentes InterSCity [de M. Del Esposte et al. 2017]. Ela é composta por quatro microsserviços[4]:

*cep-cataloger*: API REST para o cadastro de novos tipos de eventos a serem detectados e URLs que serão usadas como *web-hooks* quando as detecções ocorrerem;

*event-sender*: monitora a detecção de eventos e verifica a existência de *web-hooks* associados à detecção para dispará-los;

*cep-worker*: realiza a detecção de eventos. Cada *cep-worker* pode detectar vários tipos de eventos e é responsável por monitorar seu próprio uso de cpu e memória. Caso esteja em sobrecarga ou subcarga, ele incia um mecanismo para a realocação dos tipos de eventos que ele próprio monitora para outro *cep-worker* instanciado;

*event-composer*: compõe os dados que entram na plataforma InterSCity em tempo real e transforma-os para o formato usado pelo *cep-worker* .

No sistema, foram implementados dois algoritmos diferentes para o balanceamento de carga. O primeiro agrupa em um mesmo nó EPAs que utilizam os mesmos tipos de eventos como entrada. O segundo algoritmo mede o tamanho de estado armazenado para a detecção em cada EPA e realoca em outros nós primeiro os tipos de eventos que utilizam menos estado (para diminuir o custo da realocação).

### 3.1. Ferramentas Usadas na Implementação

Somente ferramentas de código aberto foram escolhidas para a implementação do sistema. A biblioteca de CEP usada na criação de EPAs é a **Esper**. A maioria dos artigos na área de CEP mencionam o uso do Esper, portanto essa escolha facilita a comparação do desempenho do sistema implementado com o de trabalhos relacionados.

As ferramentas **Docker**, **Kubernetes** e **Rabbitmq**[5] estão sendo usadas para facilitar a instanciação dos serviços na nuvem. Docker isola o ambiente de execução de cada uma das instâncias dos quatro microsserviços e facilita a criação e remoção de instâncias. Kubernetes é o sistema de orquestração de contêineres com maior número de *features* [Truyen et al. 2019], permitindo que o *cep-worker* tenha controle sobre sua própria terminação e sobre a criação de novas instâncias. O sistema de mensageria escolhido foi o Rabbitmq por ser o utilizado na plataforma InterSCity.

### 3.2. Avaliação

Para avaliar a capacidade de auto-escala do sistema desenvolvido, estamos implementando um experimento com a EPN da Figura 1. O experimento usa dados das posições de

[4] `https://gitlab.com/interscity/interscity-platform/{cep-worker,cep-cataloger,event-sender,event-composer}`

[5] `www.docker.com/`, `www.kubernetes.io`, `www.rabbitmq.com`

todos os ônibus de São Paulo durante seis horas contínuas, obtidos da SPTrans por meio do aplicativo Olho Vivo[6]. A EPN do experimento detecta a ocorrência de *bus bunching* no monitoramento de cada ônibus individualmente, considerando as linhas que passam pelos doze principais corredores de ônibus de São Paulo. Os dados foram coletados de um dia útil, das 6h00 da manhã até o meio-dia. Ao total, são 2360 linhas de ônibus e 26624 veículos, logo a quantidade de EPAs no experimento completo é:

- 2360 EPAs de detecção de velocidades instantânea entre 0 e 100 km/h;
- 26624 EPAs de detecção de *bus-bunching*;
- 12 EPAs de filtros de velocidade dos corredores, um EPA para cada corredor;
- 180 EPAs de detecção de velocidades baixas em ônibus nos 12 corredores.

Durante um teste preliminar com a instanciação de todos os EPAs em um só nó, executando somente o processamento de eventos, a memória RAM consumida ultrapassou 8 GB, quantidade comum de um computador médio. Esse teste evidenciou a necessidade da escalabilidade horizontal no processamento em tempo real de dados urbanos. Como próximo passo, faremos a execução do experimento em uma nuvem computacional pública. Mediremos a vazão e a latência do sistema na detecção dos eventos, bem como a variação no uso de recursos da nuvem em função da carga no sistema.

## 4. Conclusão

CEP é uma técnica poderosa para analisar dados em tempo real, mas a maioria das implementações de código aberto atuais não oferecem suporte para uma distribuição de processamento em larga escala. Escalabilidade é um requisito fundamental para lidar com dados em cidades inteligentes. Grandes cidades como São Paulo geram dados a taxas superiores às que um sistema monolítico é capaz de processar. Utilizando uma arquitetura de microsserviços para distribuir CEP, em que cada microsserviço se comunica com outros por um sistema de mensageria assíncrono, tem-se um sistema mais escalável, que permite uma alocação de recursos dinâmica. Esperamos que a ferramenta apresentada aqui facilite o desenvolvimento de aplicações para cidades inteligentes.

## Referências


de M. Del Esposte, A., Kon, F., Costa, F. M., and Lago, N. (2017). InterSCity: A scalable microservice-based open source platform for smart cities. In *Proceedings of the 6th International Conference on Smart Cities and Green ICT Systems*.

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

Freire Scattone, F. and Rosa Braghetto, K. (2018). A microservices architecture for distributed complex event processing in smart cities. In *2018 IEEE 37th International Symposium on Reliable Distributed Systems Workshops (SRDSW)*, pages 6–9.

Luckham, D. C. (2001). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Truyen, E., Landuyt, D. V., Preuveneers, D., Lagaisse, B., and Joosen, W. (2019). A comprehensive feature comparison study of open-source container orchestration frameworks. *Applied Sciences*, 9(5):931.


[6] `http://olhovivo.sptrans.com.br/`