Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

ARTHUR ALVES MELO DOS SANTOS PACHECO

MONOGRAFIA DE PROJETO ORIENTADO EM COMPUTAÇÃO I

Plataforma distribuída para detecção não supervisionada de anomalias em grandes volumes de dados

Belo Horizonte, 2º semestre de 2019

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Plataforma distribuída para detecção não supervisionada de anomalias em grandes volumes de dados

por

Arthur Alves Melo dos Santos Pacheco

Monografia de Projeto Orientado em Computação I

Apresentado como requisito da disciplina de Projeto Orientado em Computação I do Curso de Bacharelado em Ciência da Computação da UFMG

Profa. Dra. Jussara Marques de Almeida
Orientador(a)

Belo Horizonte, 2º semestre de 2019

RESUMO

O principal objetivo deste trabalho é propor uma implementação para a detecção de anomalias em tempo real para grandes volumes de dados, em particular, para bases de dados de Call Detail Records, registros gerados por empresas de telecomunicações para catalogar ações realizadas por usuários no dia a dia.

A implementação está dividida em duas partes principais: o desenvolvimento da infraestrutura paralela e distribuída utilizando componentes do ecossistema Hadoop e a detecção não supervisionada de anomalias, utilizando redes implementadas com o algoritmo Hierarchical Temporal Memory.

Este documento apresenta aspectos e desafios da computação de anomalias de dados em streaming, paradigmas utilizados no ecossistema Hadoop e um breve resumo do funcionamento e características das redes de aprendizado de Hierarchical Temporal Memory ao mesmo tempo em que propõe uma solução e apresenta resultados preliminares do funcionamento de um ambiente integrado, tanto do ponto de vista arquitetural quanto do ponto de vista de detecção de anomalias.

Palavras-chave: Hadoop, Hierarchical Temporal Memory, detecção de anomalias, CDRS

ABSTRACT

The main objective of this work is to implement real-time anomaly detection for large data volumes, in particular, for Call Detail Records databases, records generated by telecommunications companies for cataloging actions performed by users daily.

The implementation is divided into two main parts: the development of parallel and distributed infrastructure using components of the Hadoop ecosystem and unsupervised anomaly detection using networks implemented with Hierarchical Temporal Memory algorithm.

This paper presents aspects and challenges of computing streaming data anomalies, paradigms used in the Hadoop ecosystem, and a brief summary of the functioning and characteristics of Hierarchical Temporal Memory learning networks while proposing a solution and presenting preliminary results from an integrated environment.

Keywords*:* Hadoop, Hierarchical Temporal Memory, anomaly-detection, CDRS

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE SIGLAS

IoT	Internet of Things
HTM	Hierarchical Temporal Memory
CDRs	Call Detail Records
SDRs	Sparse Distributed Representations
HDFS	Hadoop Distributed File System
WORM	Write Once Read Many
HDP	Hortonworks Data Platform
API	Application Program Interface

SUMÁRIO

1 INTRODUÇÃO

Com o ritmo acelerado e crescente de geração de dados com que nos deparamos hoje (19 exabytes por mês, durante o ano de 2018, segundo dados da STATISTA[1]) em grande parte relacionado à ascensão da internet das coisas (IoT) sistemas tradicionais de computação há muito utilizados estão deixando de ser uma solução viável devido a demanda por sistemas que respondam (e se adaptem) em tempo real e pela necessidade cada vez maior da computação ser feita de maneira paralela tanto quanto possível.

As aplicações de um sistema efetivamente paralelo e adaptável perpassam as mais diversas áreas, tais como detecção de fraudes bancárias, detecção de anomalias em sistemas de telecomunicação[2], detecção de anomalias na utilização de recursos em geral[3] e até mesmo análise textual[4]. Técnicas de aprendizado supervisionado[5], em que a massa de dados é dividida em dados de treino e de dados de teste não são adequadas para esse tipo de aplicação, pois apenas uma pequena parte do volume total de dados está disponível em cada momento da computação. Técnicas não supervisionadas (em que não há necessidade de uma etapa prévia de treinamento) se tornam, portanto, fundamentais para a solução desse tipo de problema, uma vez que, os dados de entrada são massivos, não rotulados e o que é considerado um comportamento anômalo pode se alterar ao longo do tempo.

Empresas de telecomunicação geram, todos os dias, grande volumes de dados relacionados a ações que os usuários realizam em seus telefones celulares. Esses dados são denominados *Call Detail Records* (CDRs). A partir dos CDRs é possível obter uma série de informações sobre o comportamento individual de cada usuário, tais como: histórico de localização, padrões e frequência de utilização de serviços. Dado grande volume e a independência desses dados (os dados de usuários diferentes não se relacionam) faz se necessário a utilização de plataformas distribuídas e paralelas de computação e de armazenamento a fim de diminuir a latência e aumentar a vazão, características que propiciam um tempo de resposta curto, o que é fundamental para detecção e correção de anomalias em sistemas de tempo real. Uma anomalia no contexto de CDRs pode ser definida como um desvio de um padrão previamente estabelecido, tanto do ponto de vista do indivíduo (como por exemplo, alterações no padrão de chamadas realizadas) quanto do ponto de vista de infraestrutura (como, por exemplo, alterações no padrão de utilização de antenas de transmissão).

Este documento tem como objetivo apresentar uma implementação para uma solução de cunho tecnológico baseado na detecção de anomalias em CDRs, em particular na definição de padrões de utilização na duração de chamadas de voz para a detecção de comportamento anômalo.

2 FERRAMENTAS

As ferramentas utilizadas na implementação proposta se dividem em dois grupos: aprendizado e infraestrutura. No grupo de aprendizado o componente utilizado e responsável pela efetiva detecção das anomalias é o algoritmo Hierarchical Temporal Memory. Na parte de infraestrutura foram empregados uma série de componentes que compõe o Ecossistema Hadoop, propiciando assim uma maneira de realizar o processamento de forma paralela e distribuída através de um cluster de computadores.

2.1 Detecção de Anomalias

Detecção de anomalia, segundo a Encyclopedia of Database Systems: 1-5 [6], pode ser definida como a identificação de objetos incomuns em um banco de dados, ou seja, diferentes da maioria e, portanto, suspeitos resultantes de contaminação, erro ou fraude. Aplicações em streaming impõe uma série de limitações para o campo da detecção de anomalias. Tais aplicações consistem no processamento de um fluxo contínuo de dados em tempo real.

Especificando ainda mais o conceito de anomalias pode-se separá-las em duas categorias: anomalias espaciais e temporais. Anomalias espaciais são ocorrências que podem ser consideradas anômalas independente de onde ocorram, por outro lado, anomalias temporais ou contextuais levam em consideração a localização onde o dado ocorre e o contexto que culminou nessa leitura.

Na detecção de anomalias de dados em streaming a base de dados em sua completude não está disponível e a computação precisa ser feita de maneira online. Outra dificuldade apresentada por esse tipo de problema é o chamado "desvio de conceito", que é definido como uma inversão do padrão comportamental dos dados, fazendo com que um padrão anteriormente considerado anômalo se torne o novo normal. Por último, os dados em streaming geralmente são dados não rotulados, ou seja, qualquer técnica que involva a detecção de anomalias precisa aprender de maneira initerrupta e sem treinamento prévio.

No parágrafo acima foram apresentadas uma série de características particulares do procesamento de dados em tempo real e, para a detecção de anomalias nos mesmos faz-se necessária a utilização de uma ferramenta que cumpra os seguintes requisitos: seja não supervisionada, de aprendizado contínuo, que detecte tanto anomalias espaciais quanto anomalias temporais e seja sensível ao desvio de conceito. Para tanto e para fins da plataforma proposta utilizou-se uma implementação do algoritmo Hierarchical Temporal Memory.

2.2 Hierarchical Temporal Memory

Numenta é uma empresa de inteligência artificial que desenvolve uma teoria de aprendizagem baseada no neocórtex humano[7], o objetivo dos algoritmos projetados é simular de maneira fidedigna o funcionamento do neocórtex em computadores, e, a partir daí, replicar artificialmente a maneira como o cérebro adquire conhecimento. Esta teoria é conhecida como *Hierarchical Temporal Memory*, doravante denominada HTM[8]. Diferente de outros conhecidos métodos de aprendizado, tais como, redes neurais tradicionais[9], a teoria de HTM busca incorporar ao máximo aspectos biológicos do funcionamento de neurônios presentes no neocórtex.

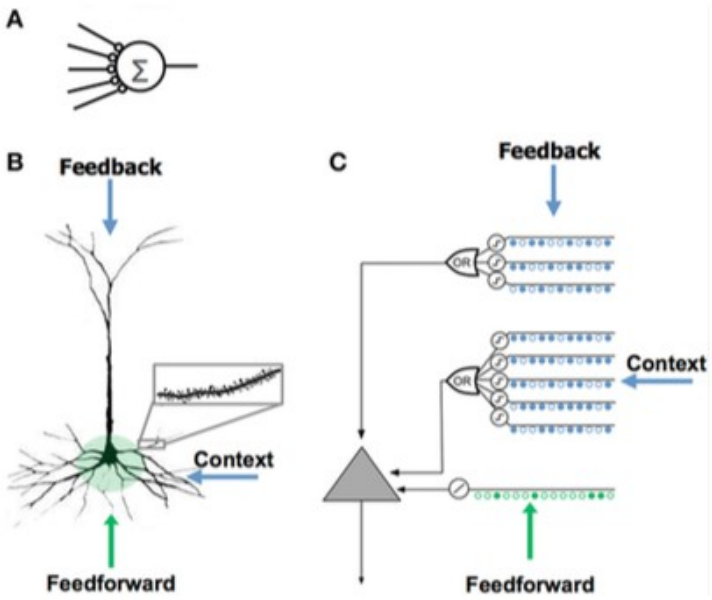


Figura 1: comparação entre neurônios de rede neurais (A), biológicos (B) e de redes HTM

A Numenta provê implementações dos módulos do HTM em Python (Nupic) e em Java (HTM Java). Neste trabalho foi utilizada apenas a API em Java[10].

O algoritmo HTM baseia-se em três grandes pilares: Sparse Distributed Representation(SDR), Spatial Pooling e Temporal Memory.

Um representação esparsa (SDR) no algoritmo da Numenta é um vetor com milhares de entradas binárias que codifica aspectos semânticos do dado. Dessa forma, as propriedades daquele objeto se tornam inerentes à sua representação. Cada bit em um SDR representa um neurônio no neocórtex. Representações esparsas parecidas possuem grande sobreposição de

bits, enquanto representaçõs distintas possuem baixa sobreposição de bits, dessa forma é possível correlacionar a ocorrência de dados similares.

As etapas de Spatial Pooling e Temporal Memory são quando o aprendizado toma forma, de fato. No Spatial Pooling o algoritmo decide quais “neurônios” possuem conexões mais fortes que satisfazem a computação daquele dado e fortalece essas conexões, criando vastas redes interconectadas, por onde as SDRs fluem. Já na etapa de Temporal Memory o algoritmo aprende padrões temporais de fluxos de SDRs. Dessa forma, ao receber um determinado input o modelo entra em modo preditivo e deixa os neurônios que são ativados com a próxima entrada prevista prontos para reagir. Quando uma previsão é confirmada a previsão daquele evento é fortalecida, do contrário ela é enfraquecida. É dessa forma que o HTM trabalha para identificar padrões temporais em uma série de dados, tais como os de utilização contidos nos CDRs.

A biblioteca Java do HTM faz uso ainda de uma API de serialização (FST Serialization[11]) para o armazenamento e recuperação do estado das redes.

2.3 Ecossistema Hadoop

Apache Hadoop é um projeto open-source para o desenvolvimento de software para computação confiável, escalável e distribuída[12]. O Hadoop permite o processamento massivo de dados através de clusters de computadores e seus principais componentes são: Hadoop Distributed File System (HDFS), YARN e MapReduce.

O Hadoop Distributed File System (HDFS) é um sistema de arquivos distribuídos, altamente tolerante a falhas e ideal para o armazenamento massivo de dados[13]. O HDFS é baseado na política WORM (write-once-read-many) e implementa nativamente redundância de dados.

O HDFS divide as máquinas em dois tipos: NameNodes e DataNodes. Os NameNodes são as peças centrais do sistema de arquivos, são eles que mantêm a árvore de diretórios e de arquivos do sistema e todas as solicitações de funções a serem performadas passam por eles. Os DataNodes por outro lado, são os pontos do sistema onde os dados são, de fato, armazenados.

O YARN[14] (acrônimo para Yet Another Resource Manager) é, em suma, um negociador. Esse módulo gerencia os recursos computacionais disponíveis entre as variadas aplicações (também conhecidas como jobs) que estão sendo executadas no cluster. É ele quem define quanto de memória e CPU serão alocados para cada tarefa e distribui conteneires de execução através das máquinas disponíveis.

O MapReduce[15] é, antes de tudo, um paradigma de computação. Esse paradigma é baseado nos conceitos de redução e mapeamento da computação, que pode assim ser feita em camadas e de maneira independente. No Hadoop ele aparece na forma de uma biblioteca interna, utilizada tanto pelo HDFS quanto pelo YARN.

Além desses, fazem parte do ecossistema Hadoop uma variedade de outros softwares que exercem papel fundamental no desenvolvimento e manutenção de sistemas para o processamento paralelo de grandes volumes de dados. Em particular, para esta implementação utilizou-se: Apache Kafka[16], Apache Storm[17] e Apache Accumulo[18].

O Apache Kafka é um sistema distribuído de streaming de dados, muito utilizado para a construção de pipelines de dados. Baseado no padrão publisher/subscriber esse sistema permite instanciar publicadores e assinantes nas estruturas denominadas tópicos, onde residem os dados, armazenados em pares key-value. O Kafka roda em um cluster de máquinas e os seus tópicos são divididos em partições, com suporte para redundância de dados. Esse módulo do ecossistema Hadoop serve como conector entre aplicações distintas, como um encanamento por onde os dados fluem.

O Apache Storm, assim como o Kafka é baseado no processamento de dados em streaming e se propõe em prover para sistemas de tempo real as funcionalidades que o MapReduce provê para sistemas batch. O Storm é baseado no conceito de topologia, que é dividida em dois componentes principais: Spout e Bolt.

Um Spout nada mais é do que um ponto de ingestão de dados. Esse componente possui integração com uma série de sistemas de armazenamento tradicionais do ecossistema Hadoop, tais como os citados anteriormente (HDFS e Kafka), nessa etapa da topologia não há muito o que ser feito com os dados de entrada além de repassá-los para um Bolt.

Nos Bolts é onde acontece a computação propriamente dita, eles são pontos da topologia onde os dados sofrem alguma transformação para então serem enviados para próximas etapas.

Os dados atravessam a topologia definida empacotados em tuplas, que podem ser enviadas individualmente ou em lotes (utilizando janelas deslizantes, por exemplo), essas tuplas são confirmadas a medida que vão passando pelos Bolts, uma vez que o Storm implementa a política at-least-once, que garante que cada tupla vai ser processada pelo menos uma vez. Ao final, caso seja necessário persistir os resultados, o Storm também fornece uma série de implementações de Bolts especiais (writers) para realizar essa tarefa.

O Apache Accumulo, por sua vez é um sistema de cache distribuído que utiliza internamente o Hadoop Distributed File System para persistir os resultados. O Accumulo é construído com base na tecnologia BigTable desenvolvida pelo Google.

Para o desenvolvimento do projeto proposto será utilizada a plataforma Hortonworks Data Platform (HDP), distribuição mantida pela empresa Cloudera que contém todos os sistemas listados em um ambiente integrado.

Todas as ferramentas detalhadas acima, em algum nível, contribuem para propiciar a infraestrutura da solução.

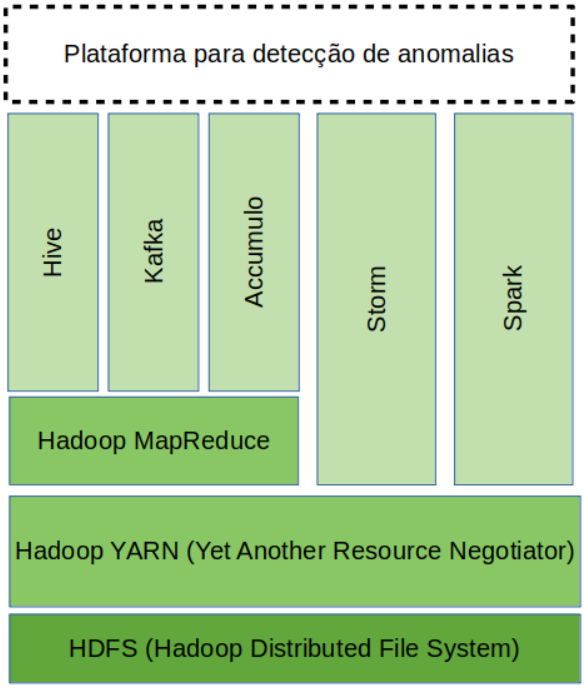


Figura 2: esquema conceitual das ferramentas utilizadas que fazem parte do ecossistema Hadoop

3 DESENVOLVIMENTO DO TRABALHO

O desenvolvimento do projeto possui quatro etapas, sendo elas:

1. Obtenção e anonimização da base de dados de CDRs.
2. Desenvolvimento do pipeline de dados utilizando as ferramentas do ecossistema Hadoop na Hortonworks Data Platform (HDP).
3. Desenvolvimento e configuração da rede HTM para o caso de uso proposto.
4. Armazenamento e análise dos resultados.

A primeira etapa consiste em ocultar dos CDRs obtidos dados sensíveis de clientes e que não são relevantes para os propósitos desse projeto, logo em seguida, na etapa de infraestrutura, o desenvolvimento da topologia Storm, bem como a configuração das ferramentas de armazenamento: Apache Kafka para armazenamento dos CDRs e Apache Accumulo para o armazenamento de redes serializadas.

A topologia desenvolvida possui cinco componentes. Em um primeiro momento os dados são recuperados de um tópico no Kafka através de um Kafka spout que recebe os pares key-value e os redistribui para o primeiro bolt. No primeiro bolt é realizado um split de todos os campos dos CDRs que são então enviados para o próximo bolt onde ocorre a seleção e agregação por usuário das features utilizadas nessa implementação, nessa etapa da topologia os campos não utilizados são descartados. A partir daí começa a computação de anomalias propriamente dita, o terceiro bolt encapsula uma fábrica de redes HTM e, uma vez recebidos os valores, prossegue-se para a criação da rede do usuário ou recuperação de rede existente no Accumulo, ao final da computação o estado da rede é salvo, serializado e retorna para o mesmo. Enquanto isso os resultados são enviados para o quarto e último bolt, onde os scores de anomalia computados são enviados novamente para um tópico no Kafka para análise posterior.

O esquema conceitual na próxima página descreve em alto nível o fluxo de dados que será implementado, desde a base de dados de CDRS até o resultado final computado pelo HTM dentro uma topologia Storm:

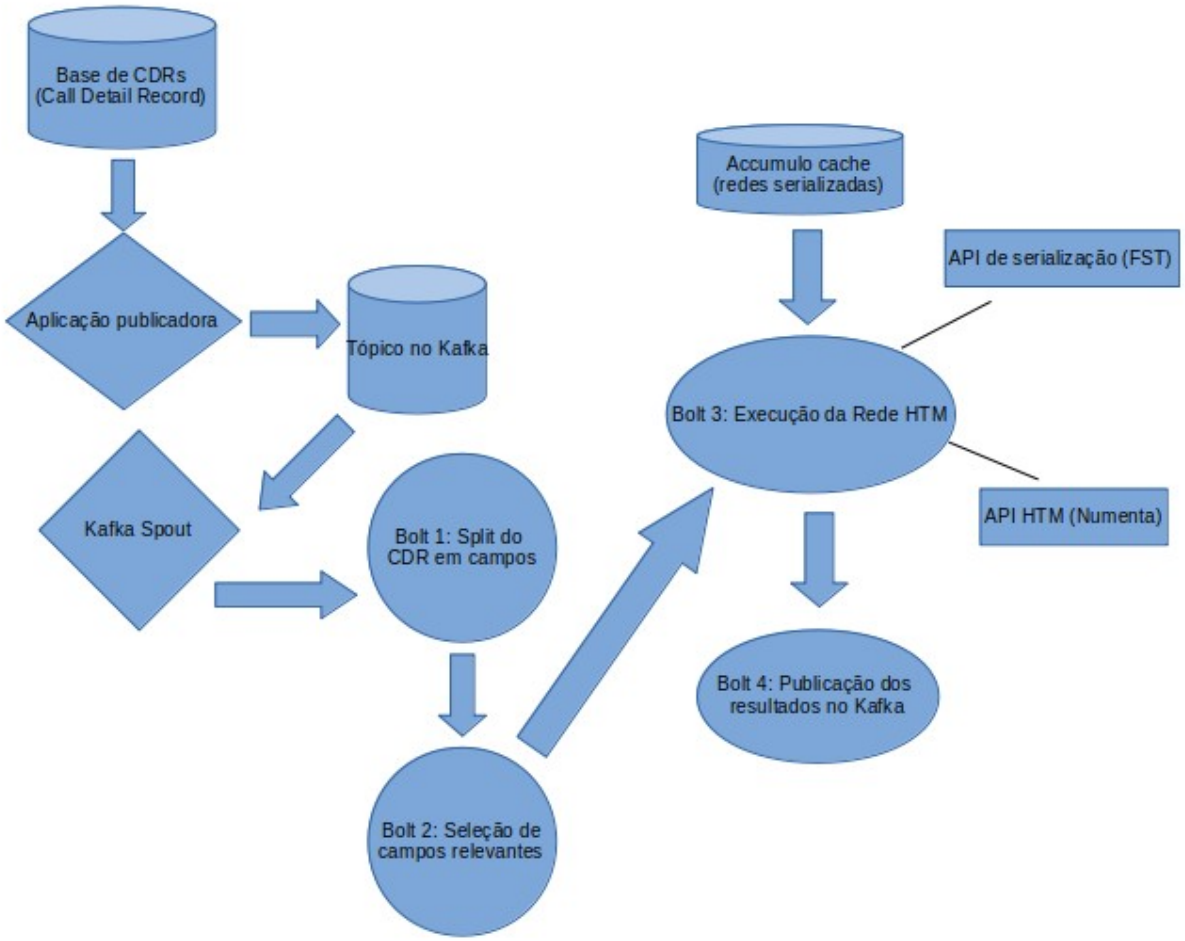
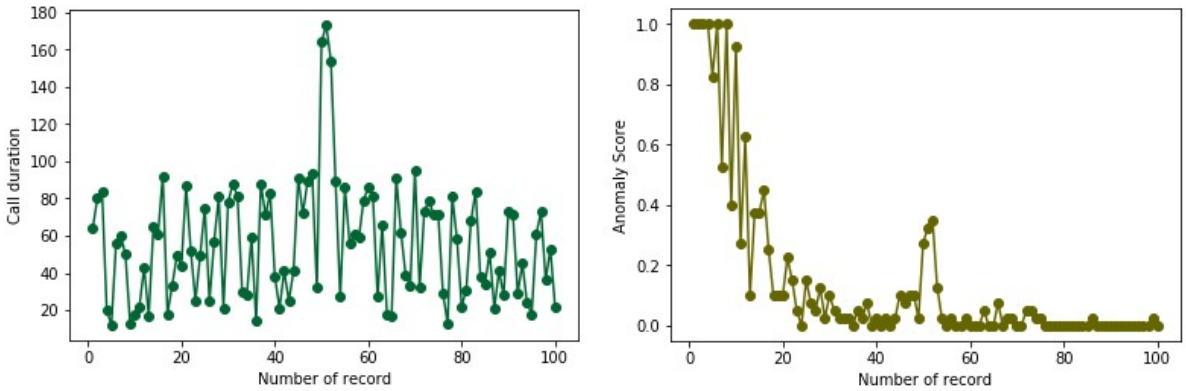


Figura 3: Descrição do fluxo de dados através da topologia Storm e da biblioteca HTM

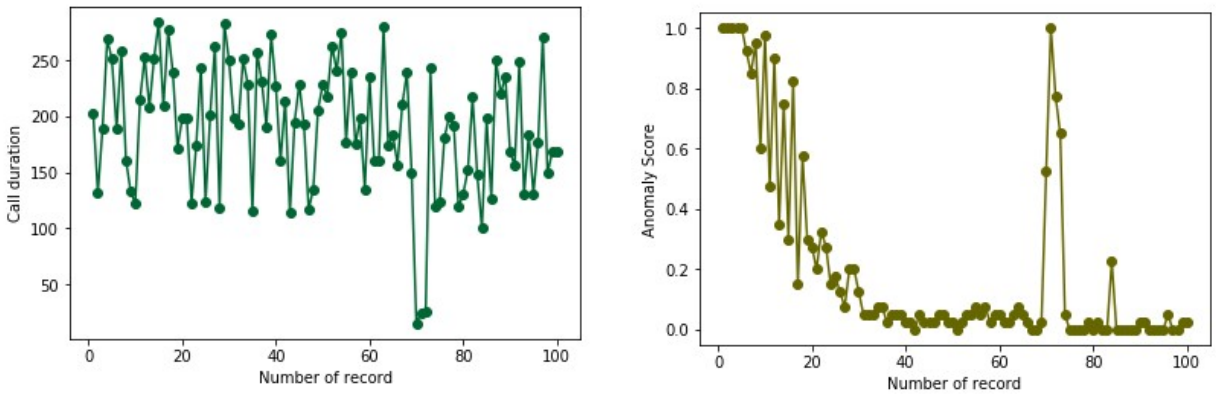
3 RESULTADOS E DISCUSSÃO

No mundo real, o protótipo a ser desenvolvido é feito para trabalhar com dados não limitados baseados em streaming, mas, para efeitos de análise dos resultados do projeto, após uma rodada de processamento a topologia foi interrompida para análise empírica dos resultados.

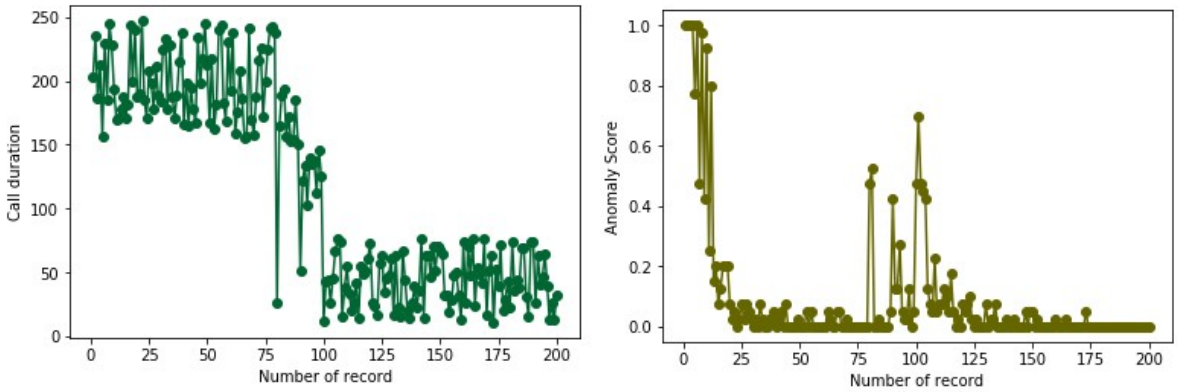
Para tal verificação foram utilizados os arquivos de log gerados pelas redes HTM armazenados no Kafka ao longo do processamento. A partir desses arquivos foram gerados os gráficos abaixo. Neles pode-se observar, a esquerda padrão de utilização de duração de chamadas ao longo do tempo de um usuário (em segundos) e à direita as respectivas pontuações de anomalia (variando de 0 a 1, onde 0 significa uma previsão perfeita e 1, registro completamente anômalo).



Gráficos 1 e 2: Exemplo de execução quando o usuário aumenta o consumo quebrando um padrão de utilização estabelecido.



Gráficos 3 e 4: Exemplo de execução quando o usuário diminui o consumo quebrando um padrão de utilização estabelecido.



Gráficos 5 e 6: Exemplo de execução quando ocorre um desvio de conceito, ou seja o usuario desenvolve um novo padrão de utilização.

Em todos os gráficos de pontuação de anomalias, observa-se que os primeiros registros são categorizados como anomalias, este fato se dá devido ao aprendizado da rede, ao começar a computar os registros de usuário, todos os dados são anômalos, pois o padrão ainda não foi aprendido, essa situação se normaliza após aproximadamente vinte e cinco registros de um padrão claro.

Dito isso, no gráfico 1 podemos observar um claro desvio de padrão no registro de número cinquenta que é prontamente identificado com um score de anomalia mais alto no seu ponto correspondente no gráfico 2. A situação se repete nos gráficos 3 e 4, onde desvios de padrão são identificados nos registros de números 71, 72 e 73.

No gráfico 5 é possível observar a ocorrência de um desvio de conceito, quando o usuário muda seu padrão de utilização, de maior consumo, para um de menor consumo. Já no gráfico 6 observa-se que de início os registros referentes ao novo padrão são identificados como anomalias, mas a medida que eles vão se tornando mais e mais comuns, a rede se adapta e passa a identificar esses valores como um novo padrão de utilização e para de reconhecê-los como anomalias.

Ao analisar os resultados, pode-se inferir, por amostragem, a eficácia da detecção de anomalias utilizando o algoritmo HTM. Foi observada também a flexibilidade da rede de se adaptar a novos padrões de dados com uma quantidade relativamente pequena de registros.

Dessa forma, o algoritmo HTM, em particular, a implementação HTM Java desenvolvida pela Numenta, foi capaz de suprir os pré-requisitos definidos na seção 2 deste documento: sem supervisão, aprendizado contínuo, detecção de anomalias temporais e espaciais e sensível ao desvio de contexto.

Código do projeto disponível em: https://github.com/arthur-pacheco/storm-topology-anomaly-detection

4 CONCLUSÕES E TRABALHOS FUTUROS

Ao implementar a solução utilizando os componentes do ecossistema Hadoop foi possível atender aos requisitos necessários a um sistema de processamento para dados massivos em tempo real. Utilizando o Kafka foi possível construir um pipeline de dados por onde os CDRs transitam e a partir daí foi definida uma topologia no Storm, onde o processamento é de fato realizado. Além do componente de infraestrutura distribuída, a utilização do framework de aprendizado Hierarchical Temporal Memory possibilitou a detecção de dados anômalos por usuário e foram apresentados alguns resultados preliminares de detecção de anomalias em uma base de dados de CDRs.

Nesta primeira parte do projeto, foi dado maior foco no desenvolvimento da infraestrutura sobre a qual a detecção de anomalias executa, na segunda parte portanto propõe-se uma avaliação mais profunda dos resultados obtidos, assim como testes de estresse, utilizando uma base de dados maior e outras configurações das redes de Hierarchical Temporal Memory para que seja definida um ponto de ótimo entre utilização de recursos computacionais e eficácia da detecção de anomalias. Outro aspecto importante de trabalhos futuros é a utilização da plataforma desenvolvida para outros casos de uso de detecção de anomalias em CDRs, como por exemplo, na utilização de estações de transmissão.

5 REFERÊNCIAS

[1] CLEMENT, J. Internet usage worldwide – Statistics & Facts. 25 de julho de 2019. Disponível em <https://www.statista.com/topics/1145/internet-usage-worldwide/>. Acesso em: 04 de dezembro de 2019

[2] Kashif Sultan, Hazrat Ali, Zhongshan Zhang. Call Detail Records Driven Anomaly Detection and Traffic Prediction in Mobile Cellular Networks, IEEE ACCESS JOURNAL, 25 July 2018, Digital Object Identifier 10.1109/ACCESS.2018.2859756

[3]Subutai Ahmada, Alexander Lavina, ScottPurdya, Zuha Aghaab. Unsupervised real-time anomaly detection for streaming data, Neurocomputing, Volume 262, 1 November 2017, Pages 134-147

[4] Cortial.IO. Página Inicial. Disponível em <https://www.cortical.io/>. Acesso em: 04 de dezembro de 2019

[5] Mohammed J. Zaki, Wagner Meira, Jr., Data Mining and Analysis: Fundamental Concepts and Algorithms, Cambridge University Press, May 2014. ISBN: 9780521766333.

[6] Zimek A., Schubert E., (2017) Outlier Detection. In: Liu L., Özsu M. (eds) Encyclopedia of Database Systems. Springer, New York, NY

[7] Numenta. Página Inicial. Disponível em <https://numenta.com/>. Acesso em: 04 de dezembro de 2010

[8] Biological and Machine Intelligence. Numenta, 8 de março de 2017. Disponível em <https://numenta.com/assets/pdf/biological-and-machine-intelligence/BAMI-Complete.pdf>. Acesso em: 04 de dezembro de 2019.

[9] Understanding Neural Networks. Towards Data Science. Disponível em <https://towardsdatascience.com/understanding-neural-networks-19020b758230>. Acesso em: 04 de dezembro de 2019

[10] HTM Java API. Disponível em <https://github.com/numenta/htm.java>, Acesso em: 04 de dezembro de 2019

[11] FST Serialization API. Disponível em <https://github.com/RuedigerMoeller/fast-serialization>. Acesso em: 04 de dezembro de 2019

[12] Hadoop. Página Inicial. Disponível em <https://hadoop.apache.org/>. Acesso em: 05 de dezembro de 2019

[13] Hadoop Documentation. HDFS Design. Disponível em <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>. Acesso em: 04 de dezembro de 2019

[14] Hadoop Documentation. YARN. Disponível em <https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>. Acesso em: 04 de dezembro de 2019

[15] ANDRADE, Thiago . MapReduce – Conceitos e Aplicações. Disponível em <http://www.ic.unicamp.br/~cortes/mo601/trabalho_mo601/tiago_cruz_map_reduce/relatorio.pdf>. Acesso em: 04 de dezembro de 2019

[16] Kafka. Intro. Disponível em <https://kafka.apache.org/intro>. Acesso em: 04 de dezembro de 2019

[17] Storm. Página Inicial. Disponível em <https://storm.apache.org/>. Acesso em: 04 de dezembro de 2019

[18] Accumulo. Página Inicial. Disponível em <https://accumulo.apache.org/>. Acesso em 04 de dezembro de 2019

[19] José Antonio Iglesias, Agapito Ledezma, Araceli Sanchis, Plamen Angelov, Real-Time Recognition of Calling Pattern and Behaviour of Mobile Phone Users through Anomaly Detection and Dynamically-Evolving Clustering, Applied Sciences, 5 August 2017

[20] Nithi and Lipika Dey, Anomaly Detection from Call Data Records, S. Chaudhury et al. (Eds.): PReMI 2009, LNCS 5909, pp. 237–242, 2009.