DATATEST: elaboração e implementação de testes automáticos em volumes gigantescos de dados

Manassés Ferreira[1], **Fabiano Nezello**[2]

[1] Diretoria de Tecnologia da Informação – Divisão de Desenvolvimento de Produtos
Universidade Federal de Minas Gerais, Belo Horizonte, Brasil

[2]CENES
Brazil

manah@ufmg.br

Abstract. *Automated warehouse testing of big data can be used to detect irregularities during the extraction, transformation and loading process. This work presents an implementation in the Apache ecosystem (Airflow, Kylin and Superset) of data-centric testing. The measures and variables involved in the tests are persisted using a dimensional model that allows the generation of historical reports. The main test implemented compares model entities through SQL queries to count rows. Tests are automated by the Apache Airflow application, the results are persisted in the Apache Kylin environment and their analysis is facilitated by dashboards/graphs generated in the Apache Superset software.*

Resumo. *Os testes automáticos em armazém (com volumes gigantescos) de dados podem ser usados para detecção de não-conformidades durante o processo de extração, transformação e carga. O presente trabalho apresenta uma implementação no ecossistema Apache (Airflow, Kylin e Superset) de testes centrados em dados. As medidas e variáveis envolvidas nos testes são persistidas em um modelo dimensional que permite a geração de relatórios históricos. O principal teste implementado compara entidades do modelo através de consultas SQL para contagem de linhas. Os testes são automatizados pela aplicação Apache Airflow, os resultados dos testes são persistidos no ambiente Apache Kylin e a análise dos mesmos é facilitada por painéis/gráficos gerados no software Apache Superset.*

Palavras-chave*: Testes automáticos; Volumes gigantescos de dados; Data warehouse; Data lakehouse; Armazém de dados; Testes do processo de ETL; Apache Airflow; Apache Kylin; Apache Superset.*

1. Introdução

A validação das informações analisadas a partir de um armazém de dados requer extrema atenção para que as tomadas de decisões não sejam viciadas negativamente por falhas críticas nesse ambiente. Essa ressalva amplia-se no contexto do armazenamento e processamento de volumes gigantescos de dados, que pode ser bem complexo, envolvendo ferramentas e tecnologias das mais diversas. Cita-se o ecossistema [Apache Kylin™] como exemplo dessa complicada arquitetura.

As informações de uma instituição quando tratadas por ferramentas de inteligência de negócio invariavelmente asseguram maior confiança e potencialidade às análises produzidas. Uma vez que tais informações são geradas em volume, velocidade e variedade crescentes, esses processos de agregação inteligente e auditável atuam exatamente para ampliar a veracidade e valor das mesmas. Por agregação inteligente destacamos, por exemplo, as técnicas para modelagem dimensional [Kimball and Ross 2011]. Os benefícios alcançados são evidentes, dentre eles ressaltamos a abstração propiciada pela análise dimensional que facilita a compreensão do negócio e a capacidade de realizar análise com base em dados históricos, o que nem sempre está disponível *per se* na aplicação em estudo. No entanto, junto da evolução das técnicas de análise dos dados, intensifica-se a necessidade de realização de testes dos mais variados tipos.

O teste em armazém de dados é bem rico, uma vez que requer avaliação de aspectos dos mais diversos [Homayouni et al. 2019]. Em um sistema de informação como esse, tão repleto de etapas e detalhes, torna-se crucial que testes guiados por processos e dados sejam realizados, preferencialmente de modo automatizado. O presente trabalho através do estudo do estado da arte em testes de armazém de dados contribui com a implementação de testes automáticos que visam avaliar o processo de extração, transformação e carga (ETL) [Gudipati et al. 2013]. Além disso, realiza a proposta de um modelo dimensional para armazenar as informações dos testes realizados produzindo assim uma base histórica para análises de evolução.

Para realizar tal empreitada adota-se a postura estratégica de estudar softwares de código aberto e de mantenedores com ilibada reputação, de inegável sucesso e com amplo suporte por parte da comunidade envolvida. No caso, as três ferramentas adotadas são desenvolvidas e mantidas pela "The Apache Software Foundation", em linhas gerais, podem ser assim descritas:

- **[Apache Airflow]** permite a automação de rotinas de trabalho genéricas por meio da linguagem de programação Python, ao disponibilizar um framework extremamente versátil, bem como uma interface gráfica para gestão de agendamentos e logs;
- **[Apache Kylin™]** (a mais complexa das três) é um motor de análise distribuído desenvolvido para prover uma interface SQL para análise em modelos dimensionais de volumes gigantescos de dados. A versão atual (v3.1.2) requer aplicações de *big data* como [Apache Spark™], [Apache™ Hadoop®], [Apache Hive™], [Apache HBase™], entre outras;
- **[Apache Superset]** é uma aplicação voltada para a exploração e visualização de dados capaz de lidar com volumes gigantescos de dados da ordem de petabytes.

Os objetivos gerais do presente trabalho foram: i) contribuir para a discussão sobre testes centrados em dados, ii) inclusive no contexto de um volume gigantesco de dados. Já os objetivos específicos foram: iii) propor testes centrados em dados interessantes para identificação de não-conformidades no contexto de extrações, transformações e cargas em um armazém de dados, iv) explorar novas tecnologias para a implementação dos testes de modo automatizado.

2. Ambiente dos ensaios

O ambiente dos ensaios pressupõe um elemento: uma base de dados já em uso, no caso, exemplifica-se usando o sistema gerenciador de banco de dados (SGBD) Oracle.

Os demais elementos envolvidos no ambiente dos ensaios foram inseridos por completo, sendo que a decisão de orquestrá-los usando ferramentas de devops (docker e docker-compose) facilitou bastante a configuração e integração dos mesmos. Essas ferramentas foram instaladas no sistema operacional linux Fedora em hardware de 24GB de memória ram e processador de oito núcleos.

Assim, em conjunto com os demais softwares brevemente descritos na seção 1, a seguinte lista: 1) Oracle; 2) Airflow; 3) Kylin e 4) Superset, resume os elementos envolvidos neste trabalho e exibidos na Figura 1 com as devidas integrações.

A Figura 1 exibe as logos das tecnologias (elementos) e aponta setas entre elas para indicar o fluxo de dados que irá ocorrer em decorrência da arquitetura determinada e das integrações viabilizadas nesse trabalho.

Figura 1. Elementos / Integrações

Os elementos da Figura 1 são integrados por meio dos seguintes middlewares[1] e com as seguintes finalidades (a serem melhor compreendidas nas seções vindouras).[2]

- **Oracle** → **Airflow** *instant_client* – o Airflow consulta o Oracle para asserções;
- **Kylin** → **Airflow** *driver jdbc* – o Airflow busca no Kylin os dados sobre o teste a ser realizado; [3]
- **Airflow** → **Kylin** *ssh + hive* – o Airflow armazena os resultados do teste no Kylin;[4]
- **Kylin** → **Superset** *kylinpy* – o Superset se conecta ao Kylin para obter os resultados dos testes.

3. Metodologia e testes propostos

O principal critério adotado para a eficiência dos testes propostos é a virtude de permitir identificar falhas (i) nos processos e (ii) nos dados de um armazém de dados. A proposta

[1]A definição principal de middleware remete ao software de computador que fornece serviços para softwares aplicativos além daqueles disponíveis pelo sistema operacional. Nesse trabalho, essa definição é adaptada de tal forma que os elementos envolvidos comportam-se como o sistema operacional.

[2]Os autores fornecem maiores detalhes sobre esses middlewares em repositório de código on-line `https://github.com/manassesferreira/datatest` e estão disponíveis para troca de comunicação eletrônica à respeito pelo e-mail informado no início do trabalho.

[3]Essa integração é útil também para o Airflow consultar o Kylin para asserções (ou seja, caso os alvos dos testes estejam armazenados em uma infraestrutura de big data).

[4]Estudos foram feitos para uso do [Apache Kafka] nessa integração e serão alvo de melhoria futura.

deste trabalho restringe-se ao estudo dos testes exibidos na Tabela 1, pertencentes às seguintes categorias: (C1) Testes do processo de ETL, (C2) Testes de unidade, (C3) Testes de integração, (C4) Testes de sistema e (C5) Testes de desempenho e escalabilidade. Essa não é uma lista de categorias exaustiva, outros aspectos poderiam ser adicionados, como por exemplo: (C6) Testes de segurança, (C7) Testes de usabilidade no processo de ETL, (C8) Testes de estresse nas aplicações front-end, entre outros.

Os testes foram levantados considerando abordagens práticas (mas não automatizadas) realizadas durante depuração de problemas / não-conformidades em processos de ETL no ambiente de trabalho do autor, bem como ideias e métodos foram obtidos através da revisão da literatura sobre o tema. O leitor pode recorrer ao trabalho [Homayouni et al. 2019] para maiores detalhes sobre esses tipos de testes.

Tabela 1. Testes e categorias

#	Descrição do teste	C1	C2	C3	C4	C5
T1	Série temporal da quantidade de linhas em uma tabela	x		x	x	
T2	Série temporal da quantidade de linhas agrupadas em uma tabela	x		x	x	
T3	Comparação da quantidade de linhas entre tabelas da origem e do destino	x		x	x	
T4	Validação na origem de colunas que violam restrição de nulidade no destino	x	x			
T5	Validação de uma lista no destino usando uma consulta à origem (verificador)	x	x	x	x	
T6	Comparação entre uma lista na origem (massa de teste) e uma lista no destino	x	x	x	x	
T7	Série temporal dos tempos de consultas (simples e complexas)				x	x
T8	Múltiplos acessos realizando consultas (simples e complexas)				x	x

Para escopo inicial do projeto adotou-se a implementação do teste T3 apenas, focando portanto em automatizar asserções sobre o processo de ETL, principalmente. O motivo reside no tempo limitado do trabalho. Além disso, boa parte desse tempo foi alocada em aprendizado das tecnologias envolvidas. Trabalhos futuros irão explorar os demais testes levantados conforme Tabela 1.

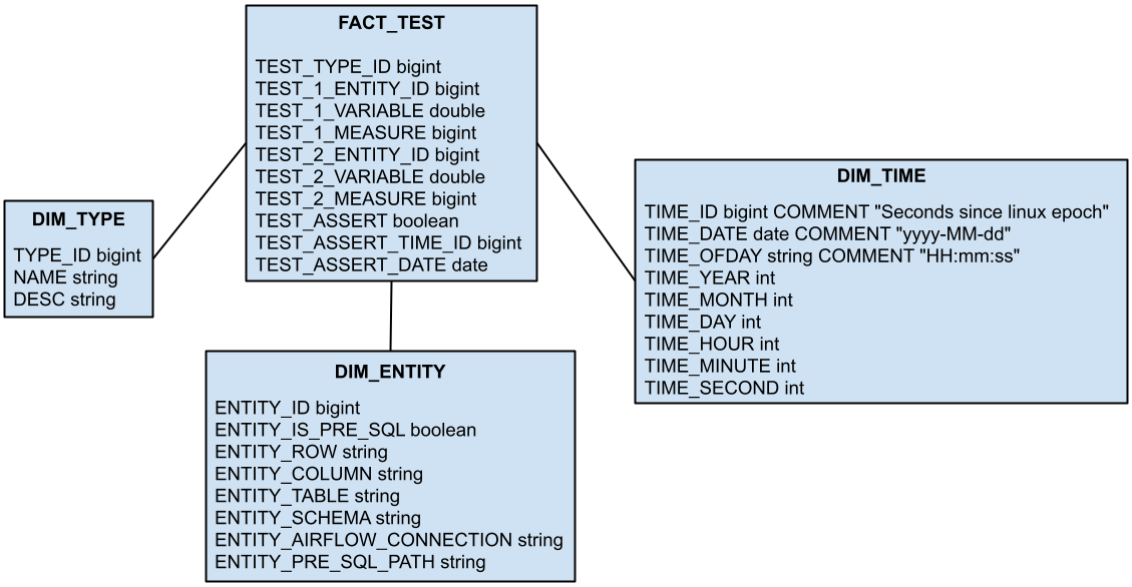
3.1. Diagrama de classes / Modelo dimensional

O diagrama de classes da Figura 2 exibe um diagrama estrela contendo um fato: o teste, e três dimensões: o tipo, a entidade e o tempo.

Os resultados dos testes propostos nesse trabalho são persistidos nesse modelo dimensional com o propósito de possibilitar: i) análises históricas (dimensão tempo DIM_TIME), ii) identificar de modo único o(s) alvo(s) (dimensão entidade DIM_ENTITY) do teste e iii) categorizar o teste (dimensão tipo DIM_TYPE). O fato em si (FACT_TEST) possui as chaves para o relacionamento com as dimensões (a saber: TEST_TYPE_ID, TEST_ASSERT_TIME_ID, TEST_1_ENTITY_ID e TEST_2_ENTITY_ID).

Observa-se que o fato foi modelado com duas entidades para permitir a comparação em pares. O fato também possui a(s) característica(s) da(s) entidade(s) em teste (TEST_1_VARIABLE e TEST_2_VARIABLE) e, os principais artefatos produzidos pelo teste, a(s) medida(s) obtida(s) (TEST_1_MEASURE e TEST_2_MEASURE) e o resultado inequívoco do teste de falha/sucesso (TEST_ASSERT).

Figura 2. Diagrama de classes / Modelo dimensional



3.2. Teste de comparação

A chave para estabelecermos discrepâncias no fluxo dos dados de um armazém de dados está, nesse tipo de teste, na comparação entre etapas (normalmente, contíguas) do processo ETL. Assim, a modelagem proposta permite comparar o número de linhas dadas duas tabelas (ou entidades, conforme o modelo dimensional proposto, descrito a seguir). Essas tabelas equiparadas serão referidas de agora em diante por Target (T) e Source (S). Finalmente, para aumentar a granularidade da comparação modela-se ainda a contraposição por meio do agrupamento das linhas para uma dada Coluna (C) da tabela. A comparação simples entre tabelas ocorre tendo por base o seguinte SQL template, exibido no Algoritmo 1.

Algoritmo 1. Comparação simples

```
SELECT COUNT(*) AS QTD
FROM <TABELA>
```

A comparação agrupada entre colunas de tabelas ocorre tendo por base o seguinte SQL template, exibido no Algoritmo 2.

Algoritmo 2. Comparação agrupada

```
SELECT <COLUNA> AS COL, COUNT(*) AS QTD
FROM <TABELA>
GROUP BY <COLUNA>
```

4. Implementação dos testes

Os testes de comparação propostos foram implementados usando a abstração grafo acíclico direcionado (em inglês, *Directed Acyclic Graph* – DAG) por meio da ferramenta [Apache Airflow]. Nesse intuito, exibe-se o Algoritmo 3 que constrói, na linguagem

python, o chamado fluxo de trabalho (ou autômato). [5]

Algoritmo 3. Fluxo de trabalho em Python para Airflow

```
from airflow import DAG
from airflow.utils.dates import days_ago, timedelta
from airflow.operators.dummy import DummyOperator
from airflow.operators.python_operator import PythonOperator
from datatest import Test

with DAG(
    'dataTest',
    default_args={'owner': 'airflow'},
    description='Enumeration and comparation data test',
    schedule_interval="*/42 * * * *",
    dagrun_timeout=timedelta(minutes=31),
    start_date=days_ago(0,0,16,0,0),
    end_date=days_ago(0,23,59,0,0),
    tags=['dataTest'],
) as dag:

    _begin = DummyOperator(task_id='begin')

    _get_test = PythonOperator(
        task_id='get_test',
        python_callable=Test.get_test,
        dag=dag
    )

    _collect_target = PythonOperator(
        task_id='collect_target',
        python_callable=Test.collect,
        op_kwargs={'fonte': 'destino' },
        dag=dag
    )

    _collect_source = PythonOperator(
        task_id='collect_source',
        python_callable=Test.collect,
        op_kwargs={'fonte': 'origem' },
        dag=dag
    )

    _assert_and_save_test = PythonOperator(
        task_id='assert_and_save_test',
        python_callable=Test._assert_and_save_test,
        dag=dag
    )

    _end = DummyOperator(
        task_id='end',
        trigger_rule="none_skipped",
    )

    _begin >> _get_test
    _get_test >> _collect_source
    _get_test >> _collect_target
    _collect_source >> _assert_and_save_test
    _collect_target >> _assert_and_save_test
    _assert_and_save_test >> _end
```

No Algoritmo 3, após as importações de bibliotecas, percebe-se que a abstração DAG inclui intervalo de agendamento de execução do autômato seguindo o padrão *cron* do unix. Exemplifica-se assim como é limpa e simples a programação do fluxo de trabalho

[5] O fluxo de trabalho corresponde ao diagrama de atividades exibido na Figura 3 da seção 4.1, adiante, onde entrada/saída do mesmo são melhor explicitadas.

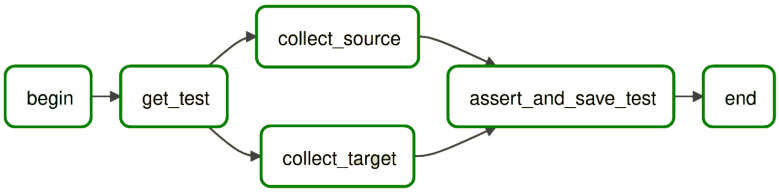
no [Apache Airflow]. Destaca-se que a biblioteca datatest foi desenvolvida nesse trabalho e está disponível no repositório `https://github.com/manassesferreira/datatest`.

4.1. Diagrama de atividades

O DAG da Figura 3 contempla os testes de comparação consistindo das seguintes tarefas, brevemente descritas.

- **get_test** busca-se o teste a ser executado. Em essência, busca-se os id's das entidades a serem comparadas e, em seguida, os demais dados dessas entidades, que permitem definir a granularidade da comparação (simples ou agrupada).
- **collect_target** coleta-se as quantidades de linhas da entidade (T).
- **collect_source** coleta-se as quantidades de linhas da entidade (S).
- **assert_and_save_test** afere-se o resultado comparando as linhas obtidas nas coletas e armazena-se o resultado e as medidas do teste.

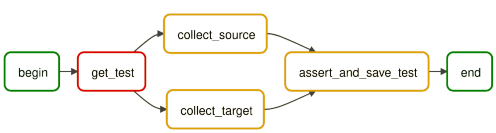
Figura 3. Diagrama de atividades / Grafo acíclico direcionado



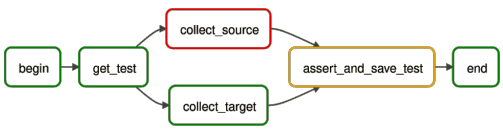
5. Resultados

Apresenta-se a Figura 4 para demonstrar brevemente as qualidades de monitoramento dos fluxos de trabalho do [Apache Airflow], inclusive lidando com falhas/exceções que podem acontecer durante as execuções. Na Figura 4a, vemos um exemplo de falha ao obter o teste (representado pela cor vermelha da tarefa **get_test**) que repercute nas etapas seguintes (**collect_target**, **collect_source** e **assert_and_save_test**, em cor amarela) dependentes dessa. De modo semelhante, na Figura 4b, vemos um exemplo de falha ao coletar os dados da fonte (S) (representado pela cor vermelha da tarefa **collect_source**) que repercute na etapa seguinte (**assert_and_save_test**, em cor amarela) dependente dessa. Para ambos exemplos, a falha não influencia na inicialização e finalização da execução do teste como um todo – o que fica evidenciado pela cor verde dos passos **begin** e **end**.

Figura 4. Falhas/exceções no fluxo de trabalho



(a) Ao obter o teste



(b) Ao coletar os dados

Para exemplificar a visualização produzida no [Apache Superset], após a execução dos testes de comparação no [Apache Airflow] e armazenamento dos resultados no [Apache Kylin™], aborda-se o confrontamento de entidades numéricas (notas

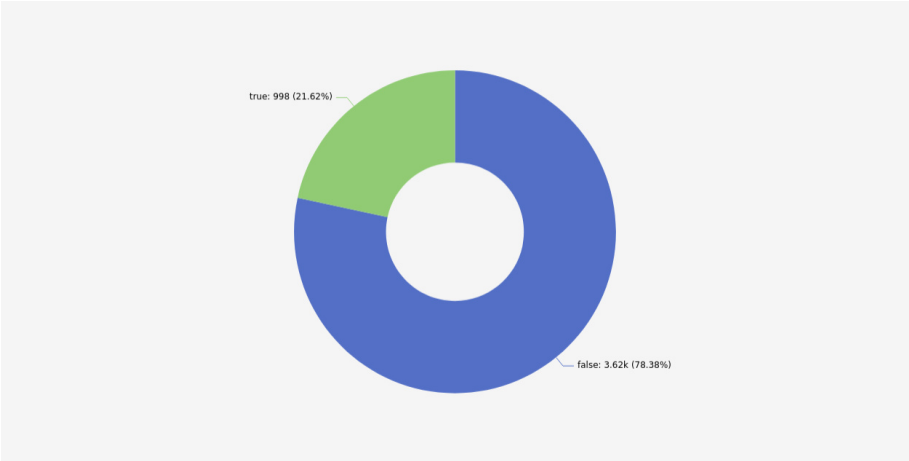
semestrais no intervalo de 0 a 100) agrupadas, tendo os resultados, que possuem não-conformidades evidentes, exibidos em três tipos de gráficos, a saber, i) Gráfico de rosca, ii) Histograma e iii) Gráfico de bolha/dispersão.

5.1. Gráfico de rosca

Para visualização do resultado como um todo do teste, mostra-se na Figura 5 que cerca de 78% das comparações apresentaram falha. No Algoritmo 4, mostra-se o template de SQL envolvido para filtrar o conjuntos de dados exibido.

O resultado esperado para um teste sem falhas seria um gráfico de rosca com 100% de *True* (sucesso).

Figura 5. Sucesso / falha do teste (Gráfico de rosca)



Algoritmo 4. Filtro dos resultados

```
SELECT TEST_ASSERT
FROM DATATEST.FACT_TEST
WHERE TEST_ASSERT_TIME_ID = 1625915746
```

5.2. Histograma

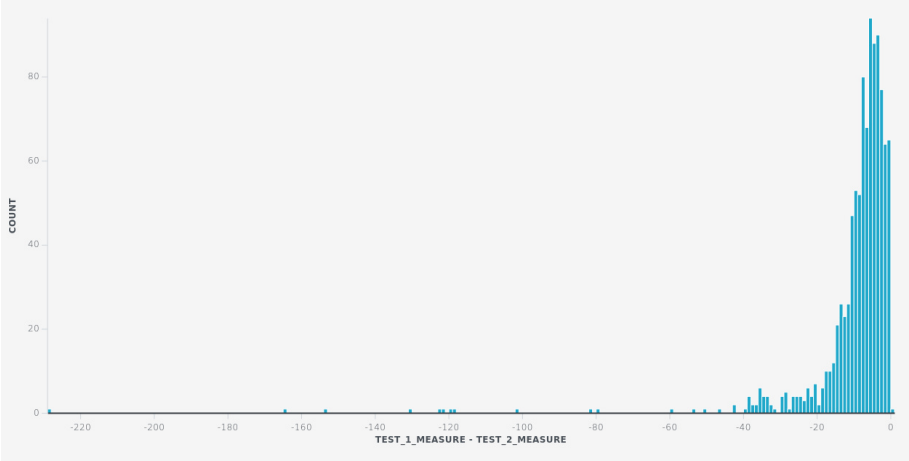
Para visualização do resultado com um maior detalhamento das falhas do teste, mostra-se na Figura 6 que desvios absolutos da ordem de até 20 unidades são aqueles que ocorrem em maior proporção, seguidos por desvios entre 20 e 40 unidades em menor proporção, bem como, fica evidente a ocorrência de desvios maiores do que 40. No Algoritmo 5, mostra-se o template de SQL envolvido para filtrar o conjuntos de dados exibido.

O resultado esperado para um teste sem falhas seria um histograma nulo, ou seja, sem desvios.

Algoritmo 5. Filtro dos resultados

```
SELECT TEST_1_MEASURE - TEST_2_MEASURE
FROM DATATEST.FACT_TEST
WHERE
   TEST_ASSERT = FALSE
   AND TEST_ASSERT_TIME_ID = 1625915746
```

Figura 6. Frequência do desvio das medidas - somente falhas (Histograma)

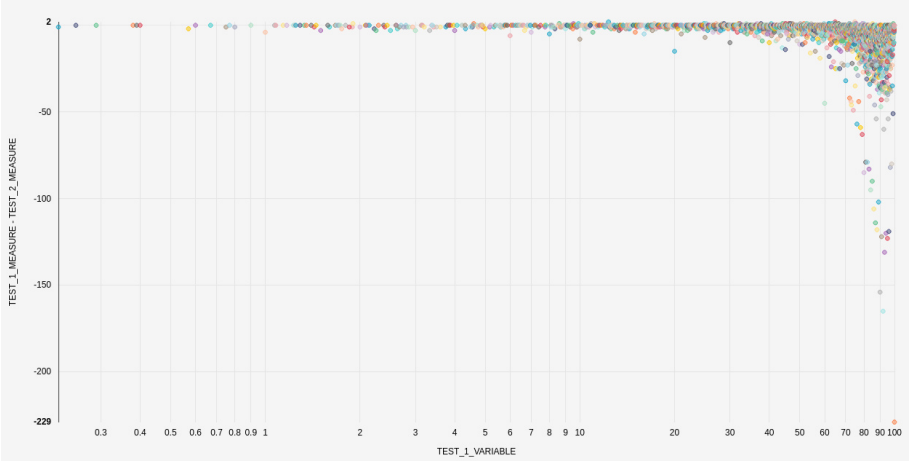


5.3. Gráfico de bolha/dispersão

Para visualização do resultado com um maior detalhamento do teste, mostra-se na Figura 7 que os maiores desvios absolutos ocorrem para as notas superiores a 50 unidades. No Algoritmo 6, mostra-se o template de SQL envolvido para filtrar o conjuntos de dados exibido.

O resultado esperado para um teste sem falhas seria um gráfico de bolha/dispersão com os pontos concentrados no eixo x.

Figura 7. Desvio das medidas por variável específica - eixo x em escala logarítimica (Gráfico de bolhas / dispersão)



Algoritmo 6. Filtro dos resultados

```
SELECT
    TEST_1_MEASURE - TEST_2_MEASURE
    ,TEST_1_VARIABLE
FROM DATATEST.FACT_TEST
WHERE TEST_ASSERT_TIME_ID = 1625915746
ORDER BY TEST_1_VARIABLE
```

6. Conclusão

Diante dos objetivos gerais e específicos, apresentados na seção 1, o presente trabalho elaborou um modelo dimensional para armazenamento de resultados dos chamados testes de comparação. Alguns estudos de caso foram feitos no contexto de processos de extração, transformação e carga em um armazém (com volume gigantesco) de dados mantido pela Diretoria de Tecnologia da Informação da Universidade Federal de Minas Gerais. Além disso, mostrou-se como automatizar tais testes usando novas tecnologias, inclusive propiciando facilidade de leitura dos resultados através de visualizações. Cabe ressaltar que as ferramentas empregadas estão preparadas para possibilitar a alta escalabilidade dos testes, tanto em termos de processamento, quanto de armazenamento.

Os resultados observados demonstraram o grande potencial da arquitetura avaliada em permitir e ancorar a implementação de testes de comparação em armazém de dados. A modelagem dimensional proposta para armazenar os resultados mostrou-se essencial para a construção de visualizações que facilitam a interpretação dos testes e permitem a caracterização das falhas. Os gráficos produzidos possuem intuito bastante geral aplicando-se até mesmo em tipos de dados não-numéricos (categóricos).

A perspectiva futura é a de que esse trabalho motive a adoção da arquitetura/modelagem apresentadas e que as mesmas acomodem naturalmente a implementação dos demais testes levantados.

Referências

Gudipati, M., Rao, S., Mohan, N. D., and Gajja, N. K. (2013). Big data: Testing approach to overcome quality challenges. *Big Data: Challenges and Opportunities*, 11(1):65–72.

Homayouni, H., Ghosh, S., and Ray, I. (2019). Data warehouse testing. In *Advances in Computers*, volume 112, pages 223–273. Elsevier.

Apache Airflow. Airflow is a platform created by the community to programmatically author, schedule and monitor workflows. `https://airflow.apache.org/`. Acessado em: 2021-05-28.

Apache HBase™. Apache hbase™ is the hadoop database, a distributed, scalable, big data store. `https://hbase.apache.org/`. Acessado em: 2021-07-17.

Apache Hive™. Apache hive™ data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using sql. structure can be projected onto data already in storage. a command line tool and jdbc driver are provided to connect users to hive. `https://hive.apache.org/`. Acessado em: 2021-07-17.

Apache Kafka. Apache kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. `https://kafka.apache.org/`. Acessado em: 2021-10-25.

Apache Kylin™. Kylin™ analytical data warehouse for big data. `http://kylin.apache.org/`. Acessado em: 2021-05-27.

Apache Spark™. Apache spark™ is a unified analytics engine for large-scale data processing. https://spark.apache.org/. Acessado em: 2021-07-17.

Apache Superset. Apache superset is a modern data exploration and visualization platform. https://superset.apache.org/. Acessado em: 2021-07-12.

Apache™ Hadoop®. Apache™ hadoop® project develops open-source software for reliable, scalable, distributed computing. https://hadoop.apache.org/. Acessado em: 2021-07-17.

Kimball, R. and Ross, M. (2011). *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons.