Um Serviço para Feedback Control Loops em Arquiteturas MapReduce

Anderson J. C. Pereira
Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBa)
Av. Araújo Pinho, nª 39 - Canela - Salvador
BA-Brasil - CEP: 40.110-150
andersoncesar@ifba.edu.br

Sandro S. Andrade
Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBa)
Av. Araújo Pinho, nª 39 - Canela - Salvador
BA-Brasil - CEP: 40.110-150
sandroandrade@ifba.edu.br

RESUMO

A análise de grandes bases de dados geradas por sistemas atuais e a execução de tarefas com demandas por alto desempenho têm sido os principais fatores motivadores do uso de tecnologias para *cloud computing*. O *MapReduce* é um estilo arquitetural e um modelo para computação distribuída amplamente utilizado atualmente e com excelente suporte à escalabilidade e tolerância a falhas de nós do *cluster*. Embora o Hadoop disponibilize uma plataforma de fácil uso para execução de *jobs MapReduce*, melhorias substanciais de desempenho são observadas ao realizar uma configuração mais especializada dos diversos parâmetros que compõem a solução. Este artigo apresenta uma extensão dos serviços básicos do Hadoop com o objetivo de suportar a execução facilitada de estratégias para *feedback control*. O objetivo é adicionar recursos de auto-gerenciamento para a otimização contínua dos parâmetros que impactam o desempenho. O artigo apresenta a arquitetura da solução proposta, um exemplo de uso de controladores PID e os resultados dos experimentos realizados.

1. INTRODUÇÃO

Nos últimos anos, a realização de operações de análise em grandes bases de dados tem sido fator determinante para o sucesso de muitas empresas. Sistemas corporativos, serviços *web* e redes sociais produzem juntos um enorme volume de dados, alcançando a dimensão de *petabytes* diários [16, 23, 14]. A realização de análises sofisticadas em bases de dados contendo informações geradas por usuários permite a investigação de tendências, derivação de perfis de uso de serviços e direcionamento de campanhas comerciais mais efetivas [22, 2].

O *MapReduce* [5] é um estilo arquitetural e um modelo computacional para processamento distribuído, amplamente utilizado na análise de *big data*. O *MapReduce* assume que os dados a serem processados estão armazenados em um sistema de arquivos distribuído e disponibiliza um *framework* que facilita mapeamento e a posterior consolidação (redução) dos dados de saída gerados. A arquitetura é inerentemente escalável e com bom suporte a falhas em nós.

O Hadoop [1, 20] é um projeto *open source* que disponibiliza duas tecnologias principais: um sistema de arquivos distribuído (HDFS – *Hadoop Distributed File System*) e uma plataforma para computação distribuída (YARN – *Yet Another Resource Negotiator*). O HDFS provê suporte escalável e tolerante a falhas para armazenamento de grandes bases de dados. O YARN suporta uma diversidade de modelos para computação paralela, dentre eles o *MapReduce*.

O desempenho de tarefas paralelas executadas sobre o Hadoop é influenciada pela natureza e organização dos dados de entrada e também por valores atribuídos a mais de 190 parâmetros, referentes ao HDFS, YARN e *MapReduce*. Embora o Hadoop defina valores *default* para estes parâmetros, de modo a viabilizar uma fácil implantação e execução destes *jobs*, estudos [3, 9, 10] demonstram que ganhos de desempenho de até 50% podem ser obtidos com valores mais especializados para as características do *cluster* e do *job* em questão. Entretanto, conhecer os pontos de configuração e os valores ótimos para um determinado *job* requer amplo conhecimento da infrastrutura e modo de operação do HDFS, do YARN e do *MapReduce*.

Tal cenário – caracterizado pela complexidade das operações de *tuning* e inexistência de uma configuração ótima para todas as situações – tem sido o foco de pesquisas relacionadas a sistemas *self-adaptive* [17]. O objetivo é dotar aplicações com a capacidade de continuamente monitorar o seu próprio funcionamento e o ambiente de execução, realizando adaptações sempre que o nível esperado de qualidade de serviço não for atendido. Como exemplos, pode-se citar os mecanismos para otimização de consumo de energia em *data centers* e os sistema de auto-ajuste de *clusters* para análise de *big data* [6, 8].

Este artigo apresenta como os serviços básicos do Hadoop podem ser estendidos de modo a suportar, de forma flexível e desacoplada, a utilização de *feedback control loops* [7] para regulação automática de parâmetros de *tuning*. O serviço de adaptação proposto permite a utilização de diferentes leis de controle e a definição de diferentes arranjos entre múltiplos *loops* de adaptação. A arquitetura da solução proposta é apresentada e discutida, em conjunto com resultados de ex-

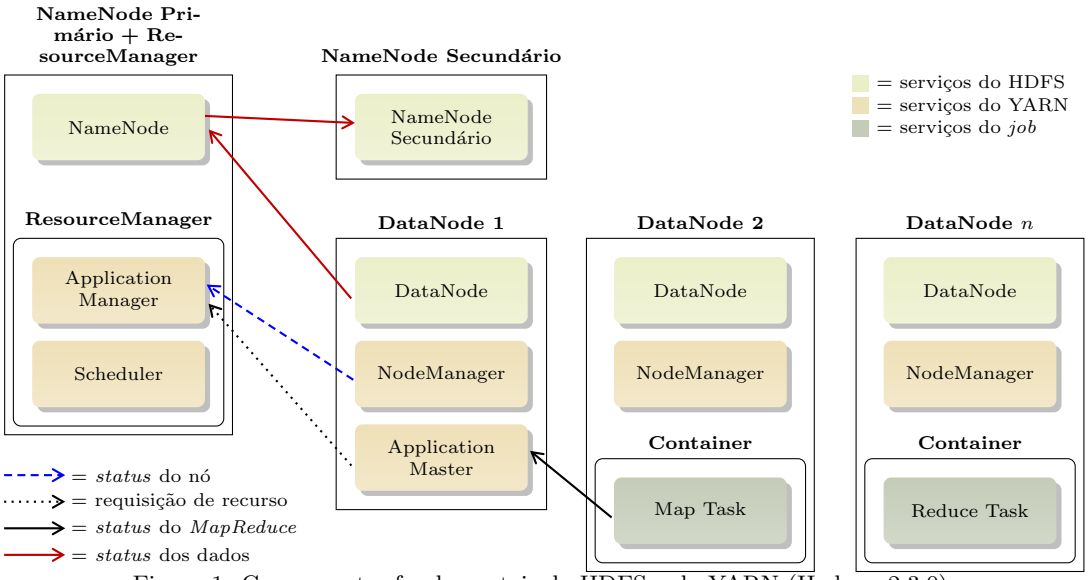

Figura 1: Componentes fundamentais do HDFS e do YARN (Hadoop 2.3.0).

perimentos de utilização do serviço de adaptação.

O restante deste artigo está organizado como segue. A seção 2 apresenta a arquitetura básica do HDFS e do YARN. A seção 3 discute os fundamentos sobre *feedback control*, enquanto a seção 4 apresenta o projeto do serviço de adaptação implementado e a 5 detalha um pouco mais sobre o processo de execução do serviço. A seção 6 apresenta o cenário de avaliação utilizado e a discussão dos resultados. Por fim, a seção 7 apresenta os trabalhos correlatos e a seção 8 conclui o artigo, apresentando possibilidades de trabalhos futuros.

2. HADOOP E APLICAÇÕES MAPREDUCE

O Hadoop2 traz uma série de decisões arquiteturais que resolvem parte dos problemas de escalabilidade identificados na sua versão anterior. A Figura 1 apresenta os principais serviços disponibilizados pelo HDFS e YARN, bem como aqueles presentes durante a execução de um *job MapReduce*.

O HDFS é formado por dois serviços básicos: o `NameNode` e o `DataNode`. O `NameNode` é responsável pelo gerenciamento global dos blocos e réplicas que representam arquivos armazenados no sistema de arquivos distribuído. É o serviço que é consultado quando deseja-se recuperar um arquivo do sistema, através da coleta e integração dos diversos blocos armazenados de forma distribuída no *cluster*. O `DataNode` é um serviço que executa em cada nó de armazenamento do *cluster* e tem como objetivo o gerenciamento dos dados que residem naquela máquina. O `DataNode` envia periodicamente ao `NameNode` informações sobre o *status* dos dados armazenados. Um `NameNode` secundário é utilizado para suportar falhas no `NameNode` primário, evitando a inviabilização de todo o *cluster*. Recursos para federação de `NameNodes`, com o objetivo de melhorar a escalabilidade do serviço, estão também disponíveis no Hadoop2.

O YARN é formado por dois serviços principais: o `ResourceManager` e o `NodeManager`. O `ResourceManager`, por sua vez, possui duas atribuições primárias: coordenar a execução de *jobs* no *cluster* e alocar recursos (memória, CPU e facilidades de comunicação) a *jobs* em execução. Tais atividades são executadas pelos componentes `ApplicationManager` e `Scheduler`, respectivamente. O `NodeManager` é um cliente do `ResourceManager`, presente em cada nó do *cluster* que está habilitado a executar tarefas de *map* ou *reduce*.

Ao receber uma solicitação de execução de *job*, o YARN seleciona uma máquina do *cluster* para hospedar a execução do `ApplicationMaster` – serviço de coordenação daquele *job* em particular. O `ApplicationMaster` é responsável pela solicitação, ao `ResourceManager`, dos recursos necessários à execução do *job*. O `Scheduler` solicita a alocação de *containers* de execução de tarefas de *map*, preferencialmente naquelas máquinas que contêm blocos do arquivo de entrada a ser processado (minimizando o tráfego de dados na rede). Dependendo do número de tarefas de *reduce* configurado para o *job*, uma ou mais máquinas do *cluster* são selecionadas para execução destas tarefas.

O Hadoop2 disponibiliza uma arquitetura flexível, onde novos escalonadores ou novos modelos de computação distribuída podem ser definidos e integrados ao *framework*. Este trabalho adiciona um novo serviço para configuração dinâmica de parâmetros do *cluster* e novos arquivos de configuração, que indicam as estratégias e parâmetros de controle a serem utilizados.

3. FEEDBACK CONTROL EM SISTEMAS COMPUTACIONAIS

Um sistema de controle [15] tem como objetivo fazer com que um determinado parâmetro de desempenho de um sistema (*measured output*) seja mantido em um valor de referência predefinido (*reference input*), através da manipulação de um valor de entrada (*control input*) que interfere no desempenho em questão. Após o desenvolvimento dos fundamentos matemáticos para estabilidade de sistemas dinâmicos no final do século XIX e das técnicas de análise e projeto de con-

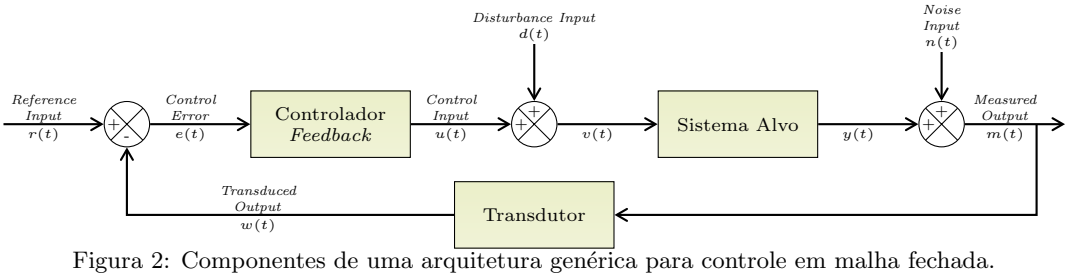

Figura 2: Componentes de uma arquitetura genérica para controle em malha fechada.

troladores no século XX, os sistemas de controle passaram a estar presentes em aplicações de diversas áreas. Atualmente, são utilizados para navegação de mísseis, aeronaves e navios; controle de níveis, temperaturas e concentrações em processos químicos; automação de plantas industriais e robôs; posicionamento de antenas; leitura de CDs; dentre inúmeras outras aplicações.

A Teoria de Controle disponibiliza todo o fundamento matemático e sistematização dos processos de projeto e análise de sistemas de controle. O objetivo é garantir que certas propriedades sejam mantidas no sistema controlado. Tais propriedades implicam, por exemplo, no conforto e segurança dos passageiros de uma aeronave, na qualidade apresentada pelo produto final de um processo químico, ou na garantia da qualidade de serviço em um sistema computacional.

A Figura 2 apresenta os principais componentes de um sistema de *feedback control*. O objetivo é monitorar uma variável de interesse (*measured output*) – como por exemplo tempo médio de resposta ou uso de CPU – e verificar quão distante esta variável está do valor para ela desejado (*reference input*). Esta diferença (*control error*) é utilizada como parâmetro de entrada para o controlador, que decide como atuar no sistema-alvo (via *Control Input*) de modo a fazer com que o *measured output* se iguale ao *reference input*.

Dentre as principais leis de controle utilizadas em sistemas com única entrada e única saída (SISO – *Single-Input Single-Output*), o PID (*Proportional-Integral-Derivative*) e suas variações têm sido amplamente utilizados na indústria. O PID decide a atuação a ser realizada no sistema a partir de três ações distintas: a primeira é proporcional ao erro atual (*control error = reference input − measured output*), a segunda é proporcional à integral do erro e a terceira, proporcional à taxa de mudança (derivada) do erro. Dessa forma podemos escrever o estado de saída do controlador PID em função de sua entrada pela equação:

$$u[k] = K_P \cdot e[k] + K_I \cdot \sum_{i=1}^{k} e[i] + K_D \cdot (e[k] - e[k-1]) \quad (1)$$

Controlador PID

Os componentes proporcional, integral ou derivativo podem ser anulados ajustando os parâmetros K_P, K_I ou K_D para zero, respectivamente, o que viabiliza a utilização de variações tais como controle P, I, PI e PD. Controladores P representam uma forma simples de compensação de perturbações, mas são conhecidos pela sua impossibilidade de redução do

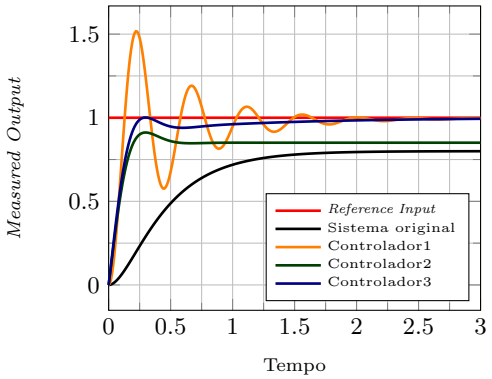

Figura 3: Ações de controladores com diferentes valores para os parâmetros K_P, K_I e K_D.

erro em regime estacionário a zero (baixa precisão). Se o erro $e[k]$ (*control error*) for zero a atuação $u[k]$ também será zero, levando à existência de erro em regime estacionário sempre que o *reference input* ou perturbação forem não-nulos.

Controladores I, por sua vez, atuam de forma proporcional à integral (histórico acumulado) do erro, ou seja, continuam ampliando a atuação mesmo quando o erro se mantém relativamente constante. Em função deste comportamento, são capazes de reduzir o erro em regime estacionário a zero. Por outro lado, este benefício é geralmente acompanhado de tempos de estabilização mais longos. Controladores PI buscam combinar a precisão dos controladores I com os tempos de estabilização mais baixos dos controladores P. Esta combinação é a mais prevalente na indústria atualmente.

O componente derivativo indica que a atuação deve ser mais forte quanto mais drástica for a mudança do erro. Embora isso ajude a projetar controladores com reação mais rápida, este comportamento mais "agressivo e precipitado"pode aumentar o *overshoot* se mal utilizado ou quando o *measured output* possui fator estocástico forte. O componente derivativo nunca é utilizado sozinho, pois um erro constante implica em derivada nula e, consequentemente, atuação nula. Controladores PD podem ser utilizados para reduzir o *overshoot* em sistemas com oscilações quando controlados via controlador P. Finalmente, controladores PID podem ser utilizados quando deseja-se combinar as influências dos três fatores.

Diferentes valores dos parâmetros K_P, K_I e K_D produzem controladores com diferentes características em relação ao tempo de estabilização e *overshoot* apresentados na resposta de controle. Conforme ilustrado na Figura 3, um sistema ori-

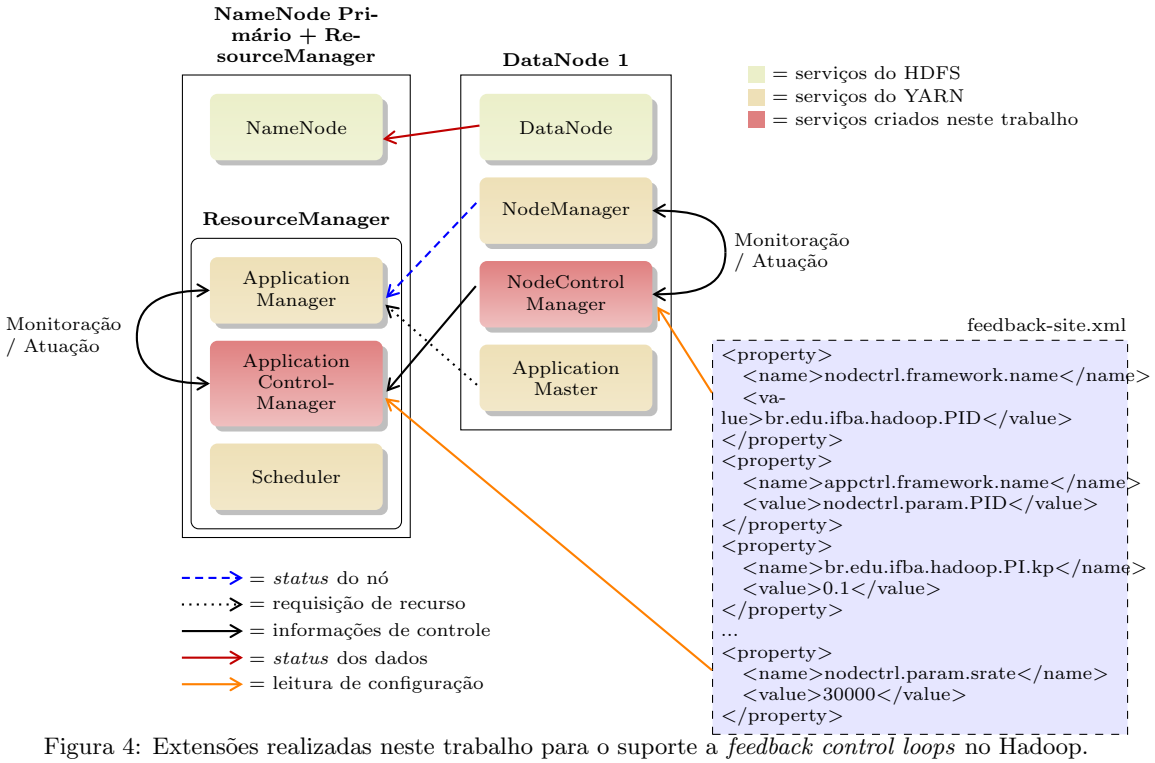

Figura 4: Extensões realizadas neste trabalho para o suporte a *feedback control loops* no Hadoop.

ginal – apresentando tempo de estabilização e erro de convergência altos – pode ter tais características melhoradas através do uso de um controlador. Embora o `controlador1` apresente erro de convergência nulo (o *measured output* passa a ser exatamente igual ao *reference input*), um alto *overshoot* é verificado. Tal situação pode implicar no uso demasiado de recursos computacionais e de rede. Já o `controlador2` apresenta *overshoot* nulo e tempo de estabilização baixo, porém a convergência é ruim (*measured output* converge para um valor menor que o *reference input*). Note como o `controlador3` apresenta um comportamento caracterizado por *overshoot* e erro de convergência nulos, associados a um baixo tempo de estabilização.

4. O SERVIÇO IMPLEMENTADO

O serviço para *feedback control loops* implementado neste artigo adiciona ao `ResourceManager` do Hadoop um suporte à configuração facilitada de estratégias específicas de controle. Conforme apresentado na Figura 4, o serviço prevê a execução de um *loop* integrado ao sistema.

O serviço `LoopManager` é responsável pela execução de adaptações locais, tais como a adição ou remoção de novos nós de processamento. O serviço é configurado a partir de um novo arquivo XML definido no Hadoop – `loopmanager-site.xml`. Neste arquivo, são informadas as estratégias de controle a serem utilizadas em cada um dos *loops*, bem como os valores de parâmetros requeridos pelo controlador sendo utilizado.

A variável a ser controlada deve ser um dos parâmetros de configuração do Hadoop e é informada através da propriedade `name` e `value` – para o controlador do `Application-Manager`. Onde `name` é o nome propriamente dito do parâmetro e o `value` é o valor inicial a ser usado. Atualmente, apenas o controlador PID e derivados estão disponíveis para uso e os serviços realizam toda a configuração dos *loops* a partir dos parâmetros descritos no arquivo `loopmanager-site.xml`. Sensores básicos para monitoração de utilização de CPU (no sistema operacional Linux) e de tempo médio de resposta de *jobs MapReduce* são também disponibilizados na solução proposta.

A compilação do Hadoop é gerenciada pela Maven [18], uma ferramenta *open source* desenvolvido pelo Apache Software Foundation, que engloba um *Project Object Model* (POM), um conjunto de normas, um sistema de gerenciamento de dependências e da lógica para a execução de metas e *plugins* em definidas fases do ciclo de vida do projeto. O Maven também pode ser usado somente como uma ferramenta de construção e de distribuição do projeto e dessa maneira foi utilizado o Maven para adicionar o nosso serviço ao Hadoop.

O *Project Object Model* (POM) ou Objeto Modelo do Projeto, possui respostas para algumas perguntas importantes sobre o projeto, como: o tipo de projeto, seu nome e as personalizações de compilação. O POM é o arquivo XML responsável pela comunicação entre o projeto e seu gerenciador, o Maven logo, para adicionar o serviço foi necessário criar o POM responsável pelas configurações e modificar alguns outros arquivos POM do projeto.

Algoritmo 1: POM do *LoopManager*

```
1
2   <parent>
3     <artifactId>hadoop-yarn-server</artifactId>
4     <groupId>org.apache.hadoop</groupId>
5     <version>3.0.0-SNAPSHOT</version>
6   </parent>
7   <modelVersion>4.0.0</modelVersion>
```

```
  <groupId>org.apache.hadoop</groupId>
  <artifactId>
  hadoop-yarn-server-loopmanager
  </artifactId>
  <version>3.0.0-SNAPSHOT</version>
  <name>hadoop-yarn-server-loopmanager</name>
  <properties>
    <yarn.basedir>
    ${project.parent.parent.basedir}
    </yarn.basedir>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-common</artifactId>
    </dependency>
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-yarn-api</artifactId>
    </dependency>
    <dependency>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
    </dependency>
  </dependencies>
</project>
```

Na imagem acima se encontram as informações necessárias para a integração do serviço de *FeedBack Control Loop* ao Hadoop. Este é um arquivo POM do Maven e está localizado na pasta do próprio serviço. O arquivo é organizado por *tags* e cada uma delas é responsável por um tipo de informação.

Para a integração do serviço ao Hadoop, é necessário informar pelo arquivo POM visto no Algoritmo 1, o POM superior que neste caso é o *hadoop-yarn-server*, responsável pelas funcionalidades de servidor do **YARN**. Essas informações se encontram no início do arquivo entre as *tags* parent. Após isso, estão as informações sobre o serviço proposto neste artigo, como a versão do POM na *tag modelVersion* e o nome do projeto na *tag name*. Por fim, no arquivo existem informações sobre dependências do projeto, que seriam o hadoop-common, um conjunto de ferramentas para todo o Hadoop, hadoop-yarn-api, API do **YARN**, e o commons-logging um conjunto de ferramentas exclusivamente para *logs*.

Com o POM criado e informando as características necessárias para que o serviço seja adicionado ao Hadoop através do Maven, é necessário ter um mecanismo de configuração mínima do serviço. Para tal questão foi criado um XML, o **loopmanager-site.xml**, como mostrado no Algoritmo 2, que possui algumas propriedades que podem ter um valor definido para melhor adequar o serviço às necessidades do usuário.

O arquivo é composto das *tags*: *configuration*, que define uma configuração que pode possuir uma ou mais propriedades, a *tag* property, que define as propriedades da configuração. Esta *tag* é composta por outras duas: *name* e *value*. A primeira descreve o nome da propriedade e a segunda o seu valor desejado. O **loopmanager-site.xml** foi baseado em outros XMLs de configuração do próprio Hadoop, como o yarn-site.xml, responsável pelas configurações do YARN e o hdfs-site.xml, responsável pelas configurações do HDFS.

Algoritmo 2: LoopManager.xml

```

<configuration>
<property>
    <name>loopmanager.time-loop</name>
    <value>140000</value>
</property>
<property>
    <name>loopmanager.setpoint</name>
    <value>50000</value>
</property>
<property>
    <name>loopmanager.kp</name>
    <value>0.01</value>
</property>
<property>
    <name>loopmanager.ki</name>
    <value>1</value>
</property>
<property>
    <name>loopmanager.kd</name>
    <value>1</value>
</property>
<property>
    <name>loopmanager.job.maps</name>
    <value>0</value>
</property>
</configuration>
```

Conforme apresentado no Algoritmo 2, a primeira propriedade é a **loopmanager.time-loop** e representa o tempo decorrido entre o início de um *loop* e seu fim. A **loopmanager.setpoint**, informa o valor que o sistema deve alcançar. As propriedades **loopmanager.kp**, **loopmanager.ki** e **loopmanager.kd**, são os representantes das variáveis proporcional, integral e derivada do controlador PID respectivamente. A última propriedade representa a configuração do Hadoop que será modificado pelo controlador PID para otimizá-lo ao valor desejado.

Para recuperar as informações do **loopmanager-site.xml** para o serviço, foi criado uma classe chamada `LoopManagerConfiguration`, uma extensão da `Configuration` do Hadoop. `Configuration` possui uma série de funcionalidades para manipulação de arquivos XML sendo utilizada para recuperar as informações dos arquivos de configuração do Hadoop, como o hdfs-site.xml. `Configuration` também possui mecanismos para encontrar o local onde estão os arquivos de configuração, pois após a compilação do Hadoop os mesmos se encontram em um diretório distinto ao de origem.

Algoritmo 3: Classe LoopManagerConfiguration.java

```
public class LoopManagerConfiguration extends
Configuration {
  @Private
  public static final String
  LOOPMANAGER_SITE_CONFIGURATION_FILE=
    "loopmanager-site.xml";

  public static final String
  LOOPMANAGER_PREFIX="loopmanager.";

  static {
    Configuration.addDefaultResource(
    LOOPMANAGER_SITE_CONFIGURATION_FILE
    );
  }

  @Private
```

```
  public static final List<String>
  LM_CONFIGURATION_FILES =
      Collections.unmodifiableList(
      Arrays.asList(
          LOOPMANAGER_SITE_CONFIGURATION_FILE
          )
          );

  public static final String TIMER_LOOP =
    LoopManagerConfiguration.LOOPMANAGER_PREFIX
        + "time-loop";

  public static final String SETPOINT =
    LoopManagerConfiguration.LOOPMANAGER_PREFIX
        + "setpoint";

  public static final String KP =
    LoopManagerConfiguration.LOOPMANAGER_PREFIX
        + "kp";

  public static final String KI =
    LoopManagerConfiguration.LOOPMANAGER_PREFIX
        + "ki";

  public static final String KD =
    LoopManagerConfiguration.LOOPMANAGER_PREFIX
        + "kd";

  public static final String JOB_MAPS =
    LoopManagerConfiguration.LOOPMANAGER_PREFIX
        + "job.maps";

  public LoopManagerConfiguration() {
    super();
  }

  public LoopManagerConfiguration(
  Configuration conf
  ) {
    super(conf);
    if (! (
    conf instanceof LoopManagerConfiguration)
        ) {
      this.reloadConfiguration();
    }
  }

}
```

A classe `LoopManagerConfiguration` possui o nome do arquivo de configuração no caso o **loopmanager-site.xml** e um método estático para cada propriedade do arquivo, que recupera o seu valor. Logo para recuperar as informações do XML se faz necessário uma instância desta classe. `LoopManagerConfiguration` segue o modelo das outras classes de configuração do Hadoop.

Para controlar todo o processo do serviço, foi criada a classe `LoopManager`. Esta classe é responsável pela recuperação dos valores no **loopmanager-site.xml**, sendo que para isso ela utiliza uma instância da classe `LoopManagerConfiguration`. Além disso `LoopManager` recupera informações sobre as tarefas *map* e/ou *reduce*, que foram executas no período do *loop*. Após essa leitura, a classe `LoopManager` utiliza o PID para descobrir se é necessário modificar a propriedade do Hadoop para alcançar a meta desejada. Ao final do seu processo, a classe grava o novo valor no arquivo de configuração da propriedade Hadoop, no caso atual o serviço vai modificar uma propriedade do *MapReduce*, logo, o arquivo a ser modificado é o mapred-site.xml.

Algoritmo 4: Método *serviceInit*

```

@Override
  protected void serviceInit(
  Configuration conf
  )
  throws Exception {
    this.conf = conf;

    this.configurationProvider =
        ConfigurationProviderFactory
        .getConfigurationProvider(conf);
    this.configurationProvider.init(this.conf);

    // load loopmanager-site.xml
    InputStream loopManagerSiteXMLInputStream =
        this.configurationProvider
        .getConfigurationInputStream(
        this.conf,
         LoopManagerConfiguration
         .LOOPMANAGER_SITE_CONFIGURATION_FILE
            );
    if (loopManagerSiteXMLInputStream != null)
        {
      this.conf.addResource(
      loopManagerSiteXMLInputStream
      );
    }

    timeLopp = Long.parseLong(
    this.conf.get(
    LoopManagerConfiguration.TIMER_LOOP
    )
    );
```

Para a recuperação dos valores do **loopmanager-site.xml** foi criado um método chamado *serviceInit*, que através da classe *ConfigurationProviderFactory*, cria uma instância da classe *ConfigurationProvider*, necessária para localizar o **loopmanager-site.xml**. A classe *InputStream* irá recuperar as informações do **loopmanager-site.xml**.

O *ResourceManager API REST* permite a obtenção de informações sobre o cluster–status no cluster, métricas no *cluster* , informações *scheduler*, informações sobre os nós do *cluster* e informações sobre aplicativos no cluster. Para a `LoopManager` as informações importantes são o *ElapsedTime*, que é o tempo de execução do *job* e o ID do *job* executado. Para tal ação usamos a classe *HttpClient* que através de uma URL recupera as informações.

Como não há a necessidade de recuperar todas as informações obtidas pelo *ResourceManager API REST*, foi necessário uma formatação dos dados obtidos e a utilização do *JavaScript Object Notation* ou Notação de Objetos *JavaScript* JSON. O JSON auxilia na formatação de dados, pois separa as informações em pares de nome/valor, assim se torna possível recuperar as informações necessárias isoladamente.

Algoritmo 5: JSON

```

JSONArray arrayJson = new JSONArray(
ResultRest.toString()
);

JSONObject json = new JSONObject(); JAVA
```

```
[rasec@rasec hadoop-3.0.0-SNAPSHOT]$ ./bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.0.0-SNAPSHOT.jar teragen 500
0 /data22/
16/03/29 09:04:16 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wh
ere applicable
16/03/29 09:04:16 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
16/03/29 09:04:17 INFO terasort.TeraSort: Generating 5000 using 2
16/03/29 09:04:17 INFO mapreduce.JobSubmitter: number of splits:2
16/03/29 09:04:17 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1459252971857_0002
16/03/29 09:04:18 INFO impl.YarnClientImpl: Submitted application application_1459252971857_0002
16/03/29 09:04:18 INFO mapreduce.Job: The url to track the job: http://rasec:8088/proxy/application_1459252971857_0002/
16/03/29 09:04:18 INFO mapreduce.Job: Running job: job_1459252971857_0002
16/03/29 09:04:24 INFO mapreduce.Job: Job job_1459252971857_0002 running in uber mode : false
16/03/29 09:04:24 INFO mapreduce.Job:  map 0% reduce 0%
16/03/29 09:04:28 INFO mapreduce.Job:  map 50% reduce 0%
16/03/29 09:04:29 INFO mapreduce.Job:  map 100% reduce 0%
16/03/29 09:04:30 INFO mapreduce.Job: Job job_1459252971857_0002 completed successfully
16/03/29 09:04:31 INFO mapreduce.Job: Counters: 31
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=239554
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=164
                HDFS: Number of bytes written=500000
                HDFS: Number of read operations=12
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
        Job Counters
                Launched map tasks=2
                Other local map tasks=2
                Total time spent by all maps in occupied slots (ms)=5828
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=5828
                Total vcore-seconds taken by all map tasks=5828
                Total megabyte-seconds taken by all map tasks=5967872
        Map-Reduce Framework
                Map input records=5000
                Map output records=5000
                Input split bytes=164
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=93
                CPU time spent (ms)=1500
                Physical memory (bytes) snapshot=345014272
                Virtual memory (bytes) snapshot=5147234304
                Total committed heap usage (bytes)=367525888
        org.apache.hadoop.examples.terasort.TeraGen$Counters
                CHECKSUM=10800304792171
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=500000
```

Figura 5: Dados do TeraGen pós execução

```
7
8  Integer elapsedTime = 0;
9
10 for (int i = 0; i < arrayJson.length(); i++) {
11
12     json = arrayJson.getJSONObject(i);
13
14     System.out.println(
15     "ID: " + json.getString("id")
16     );
17     System.out.println(
18     "ElapseTime: " + json.getInt("elapsedTime")
19     );
20     elapsedTime += json.getInt("elapsedTime");
21 }
```

Com as informações do **loopmanager-site.xml** e do *ResourceManager API REST* recuperadas, o controlador PID verifica se há um diferença entre o tempo decorrido pelo *job* *ElapsedTime* e o *SetPoint*, o tempo a ser alcançado. Atualmente está sendo usado somente o controlador PI, dessa maneira os valores do ki e do kd estão zerados.

Algoritmo 6: Controlador PID

```
1
2  double error = setPoint - elapsedTime;
3
4  errSum += (error * timeLopp);
5
6  double dErr = (error - lastErr) / timeLopp;
7  ki = 0;
8  kd = 0;
9
10 outPut = (kp * error);  + (ki * errSum) + (kd *
       dErr);
11
12 Math.round(outPut);
```

A variável error representa o erro proporcional, que é a subtração entre o tempo a ser alcançado e o tempo decorrido pelo *job*. A varável **errSum** representa o erro integral que é a multiplicação entre o erro proporcional e o tempo de repetição do *loop*. A variável **dErr** é o erro derivado é a subtração entre o erro proporcional e o último erro ocorrido no sistema dividido pelo tempo de execução do *loop*.

Ao final do processo a classe `LoopManager` grava no mapred.site.xml o novo valor para a varável de configuração. E todo esse processo é refeito quantas vezes for necessário até que o *ElapsedTime* e o *SetPoint* fiquem o mais próximo possível.

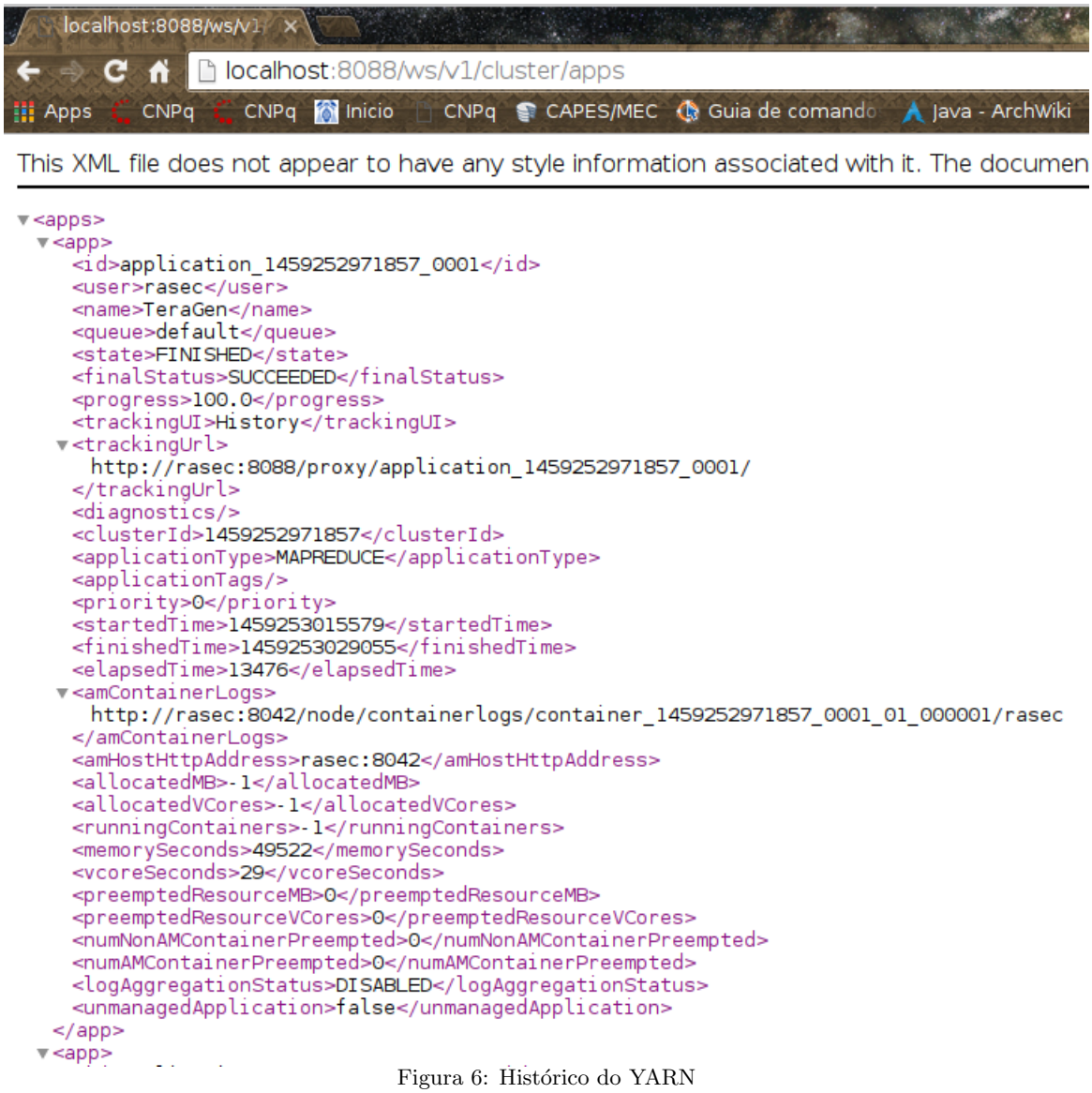

Figura 6: Histórico do YARN

5. SERVIÇO EM EXECUÇÃO

Esta seção demonstra a execução do **LoopManager** e detalha melhor sobre etapas. Como o **LoopManager** é um serviço adicionado ao Hadoop, é necessário a execução de alguns serviços do Hadoop para a utilização do **LoopManager**. Primeiro precisam ser executados os serviços do HDFS e do YARN, depois deve ser executado um *TeraGen*, após isso é possível executar o **LoopManager**.

O HDFS é responsável pelo sistema de arquivos distribuídos. O **LoopManager** precisa dele porque ao executar o *TeraGen* são gerados dados aleatórios. Além disso, um sistema de arquivos distribuídos se faz necessário para a manipulação dos dados, algo que pode ser usado no futuro.

O serviço do YARN é necessário para que o *TeraGen* seja executado, assim como o **LoopManager**, afinal ambos são parte do YARN. Como o YARN é responsável pelo processamento dos dados. Este serviço também possui um histórico dos dados que foram processados e esses dados são usados no **LoopManager**.

O *TeraGen* gera dados aleatórios no sistema de arquivos distribuídos e utiliza tarefas *maps* para organizá-los. Logo sua execução possui dois parâmetros, a quantidade de dados em *bytes* a serem gerados e o local no HDFS que serão armazenados. As informações de saída descrevem também a quantidade de tarefas *maps* utilizadas para a geração dos dados. Além disso é possível verificar o ID do *job* que está sendo executado e se foi finalizado com sucesso, ver linhas 8 e 16 da figura 5 respectivamente.

Sendo finalizada a geração dos dados no HDFS, já é possível visualizar os dados do histórico do YARN como mostrado na figura 6. Duas dessas informações são utilizadas no **LoopManager**, o ID e o *elapsedTime*. Logo o serviço implementado recupera essas informações e faz os ajustes de acordo com o necessário.

Com as informações do histórico do YARN já disponíveis,

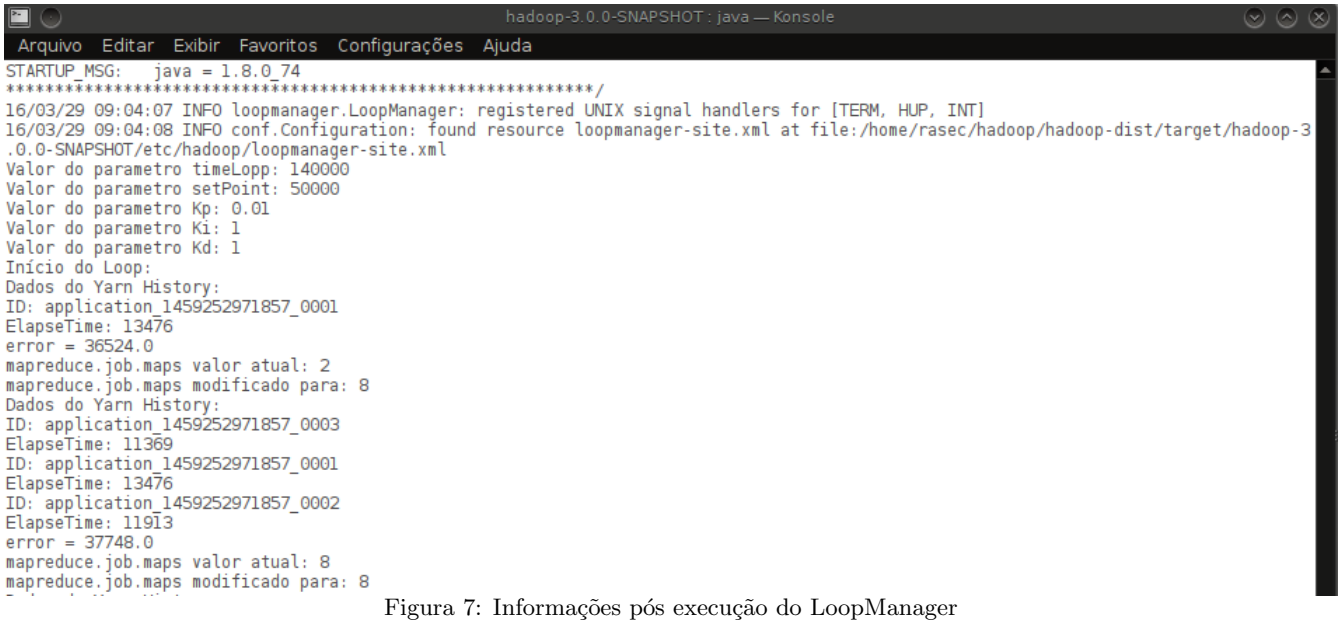

Figura 7: Informações pós execução do LoopManager

agora é possível executar o **LoopManager**. Sendo assim serão feitos todos os processos explicados no capítulo 4 referentes ao **LoopManager**, como a recuperação dos valores do histórico do YARN e o cálculo do controle PID. Ao final de todas essas etapas existem informações de saída que são disponibilizadas pelo serviço, como mostrado na figura 7.

As informações servem para mostrar os dados da configuração e do controle feito naquele *loop*. Primeiramente são descritas os dados do arquivo de configuração **LoopManager.xml**(ver o Algoritmo 2), como o *TimeLoop*, *SetPoint* e os valores para o controle PID. Após esses dados estão as informações do próprio *loop*. Essas informações podem ser modificadas a cada iteração realizada. As informações são o ID do *job*(s) que foi ou foram executados o seu tempo de execução, *ElapsedTime* e o valor final do *Error*(ver Algoritmo 6) de todo os *jobs* que foram analisados durante aquela iteração. Por último, são informados o valor atual e o modificado do parâmetro de configuração do Hadoop utilizado no **LoopManager**.

6. AVALIAÇÃO

Após a implementação do serviço como foi mostrado na seção anterior, foi realizado algumas avaliações para verificar sua funcionalidade e eficácia.

As avaliações foram realizados em um único computador, com a configuração abaixo:

- Core i7 3ª geração 2.0GHz;
- HD de 750Gb;
- 8Gb de memória RAM;
- placa gráfica Geforce 740M;
- sistema operacionl Arch Linux.

Foram feitas duas avaliações para serem comparadas e assim verificar a real eficiência do serviço. O primeiro foi realizado para descobrir o tempo gasto pelo computador para executar o *TeraGen* sem o **LoopManager**. O segundo executa o mesmo *TeraGen*, porém com o serviço em funcionamento.

Para realizar o processo de avaliação, se faz necessário iniciar os serviços HDFS e o YARN do Hadoop. O primeiro responsável por guardar as informações geradas aleatoriamente pelo *TeraGen* e o segundo é responsável pelo processamento para a formação destes dados.

A variável importante para as avaliações é a do tempo gasto para a geração dos dados *elapsedTime*, pois é ela que informa o quão otimizado está o sistema e se o mesmo se encontra perto do *setPoint*. Sempre será gerada a mesma quantidade de dados. Quando o **LoopManager** está em execução a quantidade de tarefas *map* sendo executadas é modificada automaticamente. Nas avaliações em que o **LoopManager** não está em execução será modificado manualmente a quantidade de tarefas, assim será possível verificar as discrepâncias entre as duas avaliações.

Para o primeiro experimento, a avaliação sem o **LoopManager** em execução, foi utilizado o *TeraGen* modificando manualmente a quantidade de tarefas *maps*. Como o mínimo de tarefas *maps* no Hadoop são 2 (dois) e o máximo para a configuração do computador usado para as avaliações são 8 (oito), a avaliação foi realizado nesses limites.

Dessa maneira foi gerado um *TeraGen* e após isso foi modificado o número de tarefas *maps*, partindo de 2 (dois) e indo até 8 (oito). Esse processo foi realizado 5 (cinco) vezes totalizando 35 (trinta e cinco) inicializações do *TeraGen*. As média de tempo para cada tarefa foi utilizada na comparação com o segundo teste.

No segundo experimento, o tempo de execução de um *loop* foi ajustado para um valor de 110% (cento e dez por cento)

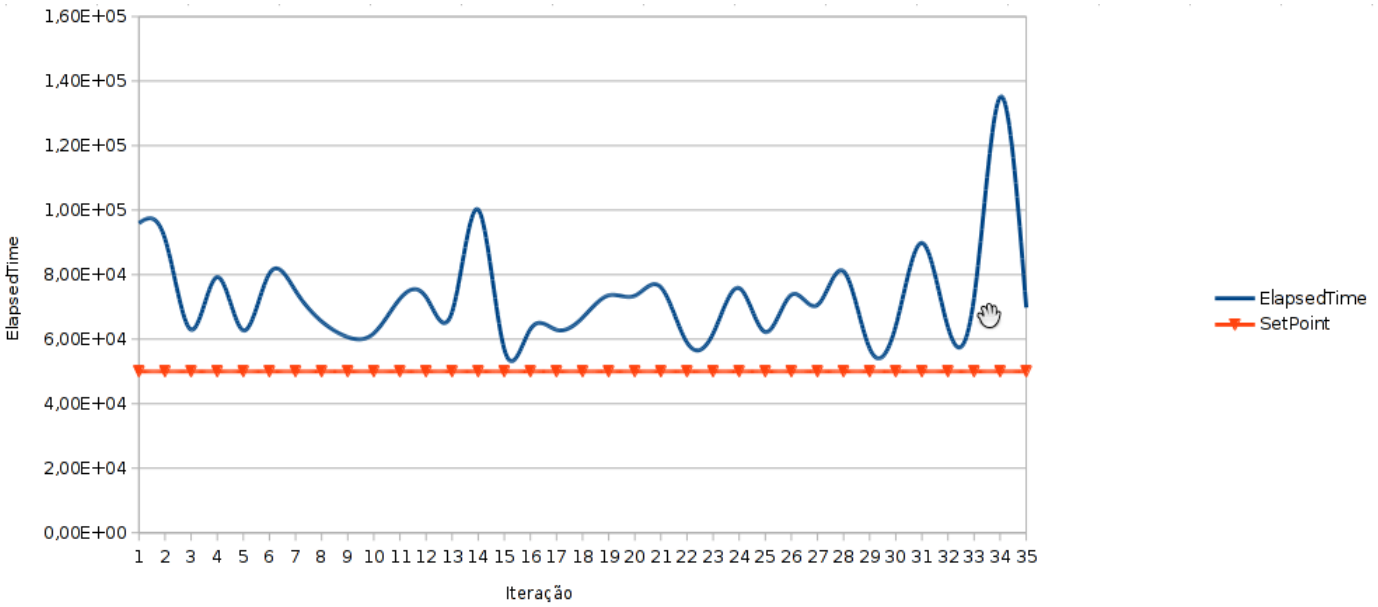

Figura 8: Gráfico Avaliação sem o LoopManager

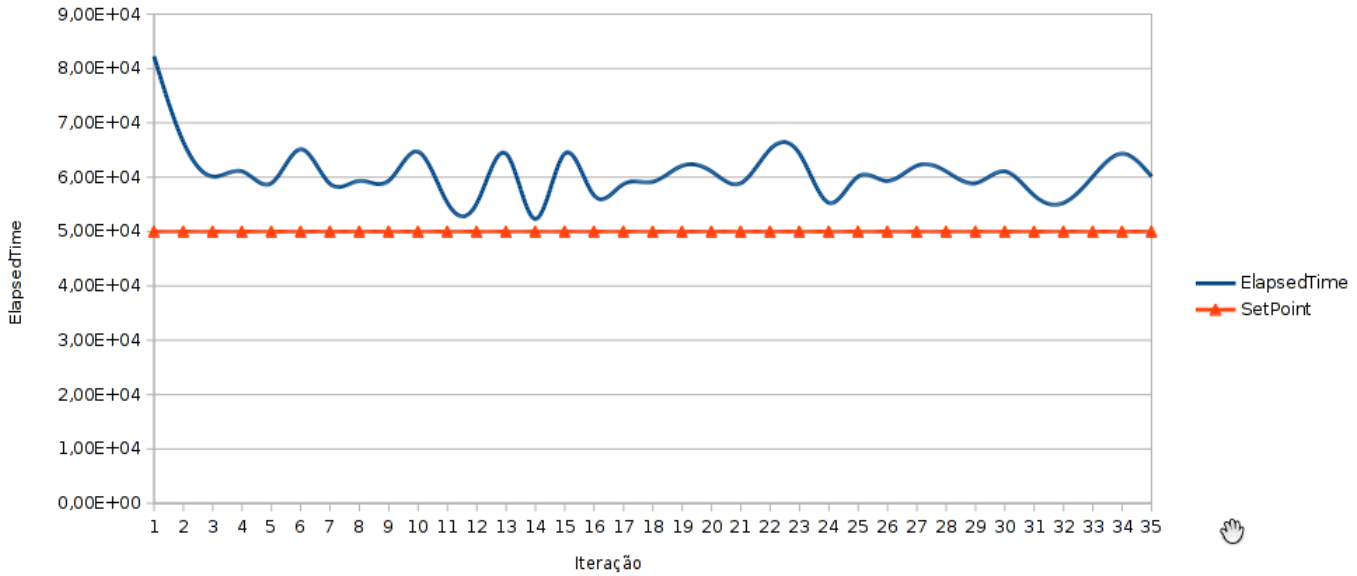

Figura 9: Gráfico Avaliação com o LoopManager

do maior tempo de todas as 35 (trinta e cinco) inicializações do *TeraGen*. O *setPoint* ou *reference input* do controlador também foi modificado para um valor entre 90% (noventa por cento) a 85% (oitenta e cinco por cento) do menor valor das iterações, logo essa é a melhoria a ser alcançada pelo sistema. Para uma melhor visualização das informações, consultar o Algoritmo: 2.

O processo do segundo teste se baseia na geração de 35 (trinta e cinco) inicializações do *TeraGen*, sendo realizados um após o outro. Dessa maneira o valor do número de tarefas será modificado, se for necessário, a cada execução.

Conforme apresentado na Figura 8 o teste sem o **LoopManager**, possui em seu eixo x a contagem do *TeraGen* e no eixo y o tempo decorrido em milissegundos. O ponto constante é o tempo a ser alcançado pelo sistema, o *SetPoint*. Os valores alcançados pelo **Hadoop** ao gerar dados através do *TeraGen* variam muito e chegam a passar do dobro do valor estipulado como meta para o sistema.

Após realizar 35 (trinta e cinco) inicializações do *TeraGen* sem o **LoopManager** o **Hadoop** não apresenta algum tipo de otimização como a do programa apresentado nesse trabalho. As oscilações no gráfico demonstram a variação do sistema em gerar a mesma quantidade de dados, e ao final do teste o tempo necessário foi muito alto chegando a quase 140.000 (cento e quarenta mil) milissegundos.

O teste realizado com o **LoopManager** está representado

na figura 9. Foram utilizadas 35 (trinta e cinco) execuções do *TeraGen*. As informações foram organizadas parecidas com o anterior, justamente para ajudar na compreensão das informações apresentadas.

É notável a diferença entre os dois gráficos, enquanto o primeiro demonstra uma certa aleatoriedade no tempo de execução, o segundo demonstra o contrário. No segundo teste, o gráfico começa com um tempo entre os 80.000 (oitenta mil) milissegundos e em poucas execuções do *TeraGen* o tempo tende a se manter em num intervalo de tempo entre 64.000 (sessenta e quatro mil) a 52.000 (cinquenta e dois mil) milissegundos.

A intervenção do **LoopManager** no **Hadoop** é visível e positiva, com apenas um parâmetro de configuração sendo modificado. Sendo assim as melhorias que podem ser realizadas através do serviço proposto aqui, podem ser mais abrangentes e efetivas.

7. TRABALHOS CORRELATOS

Uma série de esforços para implementação de comportamento *self-adaptive* em *clusters* através do uso de *feedback control* podem ser encontrados na literatura. [8] apresentam o Starfish, um serviço de auto configuração do Hadoop em três níveis: configuração de *job*, configuração de *workflow* e configuração de *workload*. A abordagem utiliza técnicas de instrumentação dinâmica e coleta de estatísticas para guiar o processo de *tuning* automático. Um novo *workflow-aware scheduler* e uma *what-if engine* são introduzidas para viabilizar a configuração de *workflows*.

[12] propõem uma abordagem que utiliza *feedback control* no gerenciamento adaptativo de máquinas virtuais. O objetivo é ajustar a utilização de recursos virtualizados de modo a alcançar os SLAs (*Service-Level Agreements*) especificados. Os autores utilizam *dynamic state space feedback control*, uma técnica para implementação de controladores MIMO (*Multiple-Inputs Multiple-Outputs*). [19] apresentam uma técnica para controle de energia em *clusters* através do uso de *model-predictive controllers*. O artigo apresenta a análise de estabilidade do controlador proposto e comparações empíricas em relação a dois outros controladores.

[13] apresentam uma abordagem para controlar o número de máquinas que devem permanecer ligadas em um *data center*, de modo a suportar o *workload* atualmente experimentado. Os autores apresentam um algoritmo chamado *Lazy Capacity Provisioning*, que utiliza uma janela de predição e um modelo ótimo de referência para decidir o número de máquinas necessárias. [4] apresentam uma solução híbrida que utiliza controladores PI e controladores *feedforward* para controlar o tempo médio de resposta de *jobs* na presença de perturbações, através da manipulação do número de nós presentes no *cluster*.

[11] apresentam uma abordagem que combina escalonadores baseados em prioridades com técnicas de *feedback control* para realizar isolamento de desempenho em aplicações *multi-tenant* em *clouds*. A abordagem baseia-se no escalonamento *Weighted Round Robin* e no uso de controladores PI. A avaliação foi realizada utilizando a plataforma *SAP Hana Cloud* em conjunto com o *benchmark* MT TPC-W. [21] apresentam uma abordagem para redução de duplicação de dados em sistemas de arquivos distribuídos através do uso de controladores. Um *Index Name Server* atua como controlador, gerenciando e otimizando os nós de armazenamento de dados de acordo com condições de transmissão dos clientes. Experimentos indicam que a abordagem diminui de 20%-50% o *overhead* de transmissão de dados devido à replicação.

A abordagem apresentada neste trabalho diferente dos estudos acima em uma série de aspectos. Primeiro, dotar o Hadoop com serviços genéricos e flexíveis para implementação de *feedback control loops* abre caminho para a avaliação facilitada de abordagens alternativas e diminui a complexidade de implementação de tais mecanismos. Segundo, o esquema de múltiplos *loops* definido neste trabalho vem sendo amplamente utilizado para controle multi-escala (em diferentes granularidades de adaptação) de serviços de *cloud computing*. Por fim, a flexibilidade de configuração das variáveis controladas e dos *inputs* de controle permite a investigação rápida dos parâmetros mais influentes e uma especificação mais produtiva dos *loops* de controle envolvidos.

8. CONCLUSÕES E TRABALHOS FUTUROS

O uso de técnicas para manutenção de qualidade de serviço em plataformas de *cloud computing* e outras infraestruturas *multi-tenant* é de fundamental importância para a diminuição de custos e boa utilização dos recursos computacionais disponíveis. O Hadoop – embora amplamente utilizado em projetos reais na indústria – carece ainda de mecanismos flexíveis e expressivos para *auto-tuning* e auto-gerenciamento em geral.

Este artigo apresentou o projeto e implementação de um serviço para utilização de *feedback control loops* em arquiteturas *MapReduce* executadas sobre o *Hadoop*. Foram apresentados os novos serviços que viabilizam a infraestrutura de controle, como eles interagem com os componentes originais do *Hadoop*, bem como aspectos de configuração que seguem as estratégias já utilizadas no *Hadoop*. Experimentos preliminares com controladores PID demonstram que um grau de satisfatório de controle pode ser obtido quando valores adequados para os parâmetros do controlador são utilizados.

Como trabalhos futuros, destaca-se: a implementação de novas estratégias de controle (ex: state-space control, escalonamento de ganho), disponibilização prévia de implementações de filtros para sensores, experimentos com *loops* em cascata e investigação de mecanismos MIMO (*Multiple-Inputs Multiple-Outputs*) de controle.

9. REFERÊNCIAS

[1] Apache Foundation. Hadoop. `http://hadoop.apache.org`. Acesso: 29/04/2014.

[2] Assunção, M. D., Calheiros, R. N., Bianchi, S., Netto, M. A., and Buyya, R. Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing 79* (2015), 3–15.

[3] Babu, S. Towards automatic optimization of mapreduce programs. In *Proceedings of the 1st ACM*

Symposium on Cloud Computing (New York, NY, USA, 2010), SoCC '10, ACM, pp. 137–142.

[4] Berekmeri, M., Serrano, D., Bouchenak, S., Marchand, N., and Robu, B. A control approach for performance of big data systems. In *IFAC World Congress* (2014). hal-00940282.

[5] Dean, J., and Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Communications of the ACM 51*, 1 (Jan. 2008), 107–113.

[6] Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S., and McKeown, N. ElasticTree: Saving energy in data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2010), NSDI'10, USENIX Association, pp. 17–17.

[7] Hellerstein, J. L., Diao, Y., Parekh, S., and Tilbury, D. M. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[8] Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B., and Babu, S. Starfish: A self-tuning system for big data analytics. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research* (2011), pp. 261–272.

[9] Jiang, D., Ooi, B. C., Shi, L., and Wu, S. The performance of MapReduce: An in-depth study. *Proceedings of the VLDB Endowment 3*, 1-2 (2010), 472–483.

[10] Kambatla, K., Pathak, A., and Pucha, H. Towards optimizing Hadoop provisioning in the cloud. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing* (Berkeley, CA, USA, 2009), HotCloud'09, USENIX Association.

[11] Krebs, R., and Mehta, A. A feedback controlled scheduler for performance isolation in multi-tenant applications. In *Cloud and Green Computing (CGC), 2013 Third International Conference on* (2013), IEEE, pp. 195–196.

[12] Li, Q., Hao, Q., Xiao, L., and Li, Z. Adaptive management of virtualized resources in cloud computing using feedback control. In *Information Science and Engineering (ICISE), 2009 1st International Conference on* (2009), IEEE, pp. 99–102.

[13] Lin, M., Wierman, A., Andrew, L. L., and Thereska, E. Dynamic right-sizing for power-proportional data centers. *Networking, IEEE/ACM Transactions on 21*, 5 (2013), 1378–1391.

[14] Marz, N., and Warren, J. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, 1st ed. Manning Publications Co., Greenwich, CT, USA, 2015.

[15] Ogata, K. *Modern Control Engineering (5th Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[16] Russom, P., et al. Big data analytics. *TDWI Best Practices Report, Fourth Quarter* (2011).

[17] Salehie, M., and Tahvildari, L. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems 4*, 2 (May 2009), 14:1–14:42.

[18] Van Zyl, J., Fox, B., Casey, J., Snyder, B., O'Brien, T., and Redmond, E. Maven: The definitive guide. *First.[[O'Reilly Media]]* (2008).

[19] Wang, X., and Chen, M. Cluster-level feedback power control for performance optimization. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on* (2008), IEEE, pp. 101–110.

[20] White, T. *Hadoop: The Definitive Guide*, 3rd ed. O'Reilly Media, Inc., 2012.

[21] Wu, T.-Y., Pan, J.-S., and Lin, C.-F. Improving accessing efficiency of cloud storage using de-duplication and feedback schemes. *IEEE Systems Journal 8*, 1 (2014), 208–218.

[22] Zhang, Q., Cheng, L., and Boutaba, R. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications 1*, 1 (2010), 7–18.

[23] Zikopoulos, P., Eaton, C., et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.