UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DE UM ESCALONADOR SENSÍVEL AO CONTEXTO PARA O APACHE HADOOP

TRABALHO DE GRADUAÇÃO

Guilherme Weigert Cassales

Santa Maria, RS, Brasil

2013

DESENVOLVIMENTO DE UM ESCALONADOR SENSÍVEL AO CONTEXTO PARA O APACHE HADOOP

Guilherme Weigert Cassales

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Ciência da Computação

Orientadora: Profª. Drª. Andrea Schwertner Charão

Santa Maria, RS, Brasil

2013

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

DESENVOLVIMENTO DE UM ESCALONADOR SENSÍVEL AO CONTEXTO PARA O APACHE HADOOP

elaborado por
Guilherme Weigert Cassales

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Andrea Schwertner Charão, Drª.
(Presidente/Orientadora)

Benhur de Oliveira Stein, Prof. Dr. (UFSM)

Patrícia Pitthan de Araújo Barcelos, Profª. Drª. (UFSM)

Santa Maria, 13 de Novembro de 2013.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

DESENVOLVIMENTO DE UM ESCALONADOR SENSÍVEL AO CONTEXTO PARA O APACHE HADOOP

AUTOR: GUILHERME WEIGERT CASSALES
ORIENTADORA: ANDREA SCHWERTNER CHARÃO
Local da Defesa e Data: Santa Maria, 13 de Novembro de 2013.

Hoje em dia, o volume de dados gerados é muito maior do que a capacidade de processamento dos computadores. Como solução para esse problema, algumas tarefas podem ser paralelizadas ou distribuidas. O *framework Apache Hadoop* (**?**), é uma delas e poupa o programador as terefas de gerenciamento, como tolerância à falhas, particionamento dos dados entre outros. Um problema no escalonador do *Apache Hadoop* é que seu foco é em ambientes homogêneos, o que muitas vezes não é possível de se manter. O foco deste trabalho é implementar um novo escalonador que seja sensível ao contexto e que leve em conta as capacidades físicas de cada máquina na hora de distribuir as tarefas submetidas.

Nowadays the volume of data generated by the services provided for end users, is way larger than the processing capacity of one computer alone. As a solution to this problem, some tasks can be parallelized. The Apache Hadoop framework (**?**), is one of these parallelized solutions and it spares the programmer of management tasks such as fault tolerance, data partitioning, among others. One problem on this framework is the scheduler, which is designed for homogeneous environments. It is worth to remember that maintaining a homogeneous environment is somewhat difficult today, given the fast development of new, cheaper and more powerful hardware. This work focuses on altering the Capacity Scheduler, in order to make it more context-aware towards resources on the cluster since, the original scheduler assumes that every node on cluster has the same amount of memory and therefore may set the resources too high or too low. This work managed to insert code on NodeManager methods so that it will collect data on every node and send it to CapacityScheduler, which will adjust the allocation values according to the total cluster resources.

Palavras-chave: Apache Hadoop. Escalonador. Sensibilidade ao Contexto.

SUMÁRIO

1 INTRODUÇÃO

Uma das maiores empresas de tecnologia da atualidade, o Google (**?**), teve a ideia inicial de uma maneira de processar o grande volume de dados que era gerado em seus servidores. Tratava-se de uma abordagem que posteriormente ficaria conhecida como *MapReduce*, constituída de duas etapas, cada uma utilizando uma função já conhecida das linguagens funcionais.

Paralelamente a isso, um projeto liderado pela Yahoo! (**?**) também começava uma implementação de *MapReduce* para seu sistema, que mais tarde tornou-se um projeto separado conhecido como *Apache Hadoop* (**?**).

Hoje o *framework Apache Hadoop* possui uma comunidade muito ativa tanto de desenvolvedores como usuários, porém algumas de suas características da época de sua criação não foram alteradas, entre elas o seu foco em ambientes homogêneos. Sabe-se que manter um ambiente totalmente homogêneo torna-se complicado com o passar do tempo, pois é comum a substituição de peças e de máquinas defeituosas por componentes mais atuais.

O desempenho das tarefas de *MapReduce* está fortemente atrelada ao escalonador do Hadoop(**?**). Por tratar-se de um projeto de *open-source*, é possível criar um novo escalonador que se adapte e faça uso da heterogeneidade do ambiente de forma a melhorar o desempenho.

Uma característica chave para adaptação do *framework* Hadoop para ambientes heterogêneos é a sensibilidade ao contexto. A definição do contexto pode variar de uma aplicação para outra, mas de maneira geral é uma informação que a aplicação irá utilizar como base para a tomada de suas decisões. Quando uma aplicação é sensível ao contexto, ela irá detectar e adaptar-se às mudanças que ocorram no ambiente. (**?**).

No âmbito do presente trabalho, o contexto ao qual a aplicação deverá se adaptar são as configurações físicas das máquinas que compõem um *cluster* Hadoop, permitindo às tarefas que demandem maior quantidade de recursos sejam executadas em máquinas mais potentes do *cluster*. Num grau de complexidade maior, o qual o presente trabalho está inserido, existe um projeto (**?**) que tem por objetivo adaptar-se a mais variações do ambiente, como a inserção e remoção de nós em tempo de execução.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é o desenvolvimento de um novo escalonador sensível ao contexto para o *Apache Hadoop*, tornando este *framework* capaz de adaptar-se a ambientes de execução heterogêneos.

1.1.2 Objetivos Específicos

- Estudar os *schedulers* do *Apache Hadoop*.
- Desenvolvimento de um escalonador base.
- Estudo da relação das Máquinas Virtuais(MVs) com o hardware da máquina.
- Desenvolvimento de um novo escalonador que utilize as informações das máquinas físicas.
- Testes do novo escalonador.

1.2 Justificativa

Atualmente, algumas tarefas de processamento que outrora eram destinadas a grandes *mainframes* e servidores está gradualmente passando para *clusters* de computadores de preços acessíveis e que são facilmente obtidos no mercado. Como prova disso, está a crescente utilização de *frameworks* em *clusters*, sendo que o *Apache Hadoop* também segue essa linha.

Apesar de ter como alvo os *clusters*, o *Apache Hadoop* foi implementado sobre a suposição de que todos os nós em um *cluster* seriam homogêneos, fazendo desse um requisito que muitas vezes se torna complicado de ser satisfeito dentro de um *cluster*.

O presente trabalho é relevante, pois seu objetivo está baseado na adaptação e melhoria de uma tecnologia já existente. Com o desenvolvimento de um novo escalonador, não só os *clusters* utilizadores do *Apache Hadoop* terão uma maneira de aproveitar melhor suas máquinas, como o próprio *framework* tornar-se-á mais versátil e capaz de adaptar-se a ambientes heterogêneos.

2 FUNDAMENTOS E REVISÃO DE LITERATURA

Este capítulo destina-se à definição de conceitos teóricos sobre as ferramentas e paradigmas utilizados no trabalho, os quais são listados a seguir: *Framework Apache Hadoop*, *MapReduce*, bem como trabalhos relacionados.

2.1 Hadoop

A origem do *framework Apache Hadoop*, vem de outro projeto da *Apache* (**?**), o *Apache Nutch* (**?**), que era um motor de buscas na *web* com código livre iniciado em 2002. Porém o projeto encontrava problemas devido a sua arquitetura. Em 2003 quando a *Google* publicou um artigo descrevendo a arquitetura utilizado no seu sistema de arquivos distribuídos, chamado GFS, os desenvolvedores viram que uma arquitetura semelhante resolveria o problema de escalabilidade do *Nutch*.

Em 2004 os desenvolvedores do *Nutch* começaram a implementar a ideia e o resultado foi nomeado *Nutch Distributed Filesystem* (NDFS). A medida que o projeto avançava ele foi tomando proporções cada vez maiores, até que em 2006 foi criado um novo projeto pois os avanços já ultrapassavam o propósito do *Nutch*, o novo projeto foi nomeado *Hadoop*. O *framework Hadoop* tem o propósito de facilitar o processamento distribuído através do paradigma do *MapReduce*.

2.1.1 Arquitetura geral do *Apache Hadoop*

De maneira geral é possível separar o *Apache Hadoop* em duas partes, as quais são denominadas *HDFS (Hadoop Distributed File System)* e *YARN (Yet Another Resource Negotiator)*. A Figura **??** fornece uma visão de como o *framework* é estruturado.

O HDFS é a parte responsável pelo armazenamento dos dados necessários para que os jobs sejam executados, em outras palavras, é um grande HD distribuído como indica sua denominação (*Distributed File System*). O HDFS é o componente que irá fazer a replicação de tolerância a falhas, distribuição dos dados de acordo com o que cada nó irá processar, entre outras atribuições.

A outra metade do *Apache Hadoop*, o YARN, é responsável pelo processamento dos *jobs* submetidos ao *cluster*. É dentro do YARN que as tarefas de *MapReduce* são executadas,

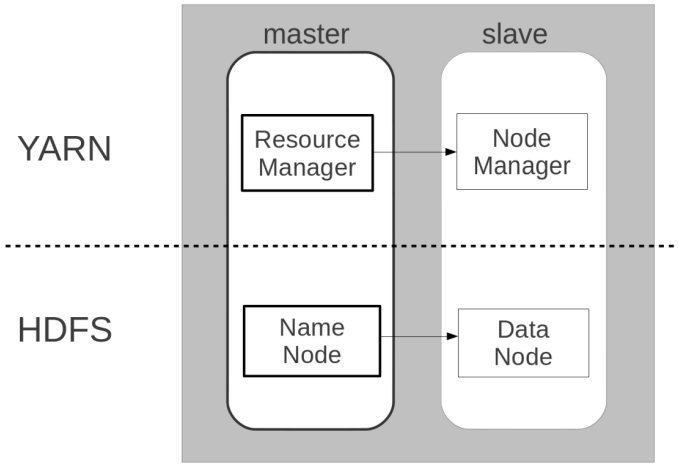

Figura 2.1 – Arquitetura geral do *Apache Hadoop*

consequentemente o YARN é o componente que gerencia todos os recursos do *cluster*.

2.1.1.1 HDFS

O HDFS é em grande parte responsável pelo bom desempenho do *Apache Hadoop*, pois é encarregado com a tarefa de não sobrecarregar a rede com transferência de arquivos. No HDFS o acesso a arquivo é sempre local, isso quer dizer que cada nó receberá a parte do arquivo referente a sua carga de trabalho, evitando assim replicação desnecessária além da básica para segurança e tolerância a falhas. Um problema dessa abordagem é que o *Hadoop* possui uma latência muito alta, sendo desaconselhável o uso do *Hadoop* em aplicações críticas ou de tempo real. O HDFS pode ser subdivido em dois serviços, *NameNode* e *DataNode*, responsáveis pelo gerenciamento dos dados a nível de *cluster* e gerenciamento dos dados a nível local, respectivamente. A Figura **??** apresenta um esquema básico da arquitetura do HDFS.

2.1.1.2 YARN

O YARN é a parte do *Apache Hadoop* responsável pela execução do *MapReduce*, portanto à ele cabem as tarefas de gerenciamento e execução do processamento. Ao tornar a tarefa de processamento totalmente independente das tarefas de armazenamento, o *Apache Hadoop* abre muitas possibilidades para sua utilização. Assim como o HDFS, o YARN pode ser subdividido em 2 serviços, *ResourceManager* e *NodeManager*, responsáveis pelo gerenciamento dos recursos no sistema e pelo gerenciamento dos recursos locais, respectivamente. A Figura **??** apresenta um esquema básico da arquitetura do YARN.

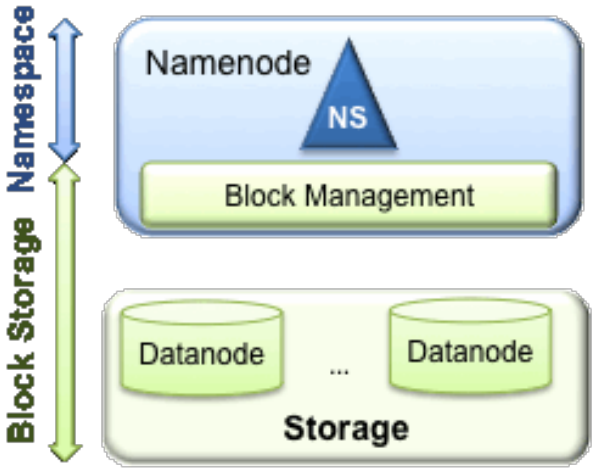

Figura 2.2 – Arquitetura geral do HDFS (**?**)

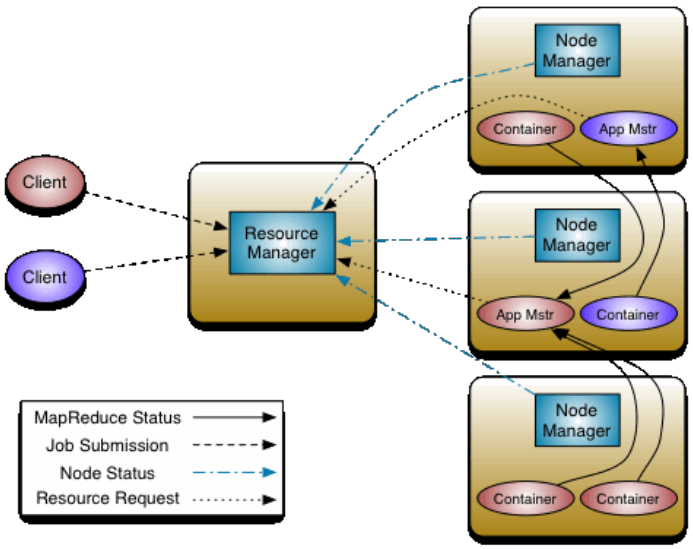

Figura 2.3 – Arquitetura geral do YARN (**?**)

Embora não demonstrado na imagem, cada serviço possui diversos módulos internos, por exemplo o *ResourceManager* possui o Escalonador e o *ApplicationsManager*, os quais ainda podem ser divididos em sub-módulos menores. O presente trabalho busca apresentar uma nova solução para a maneira de escalonamento empregada no *Hadoop* que se adapte melhor ao ambiente.

2.1.2 Configuração do ambiente de execução do *Hadoop*

Um ambiente corretamente configurado do *Hadoop* possui alguns pré-requisitos além dos nós acessíveis entre si em uma rede. Cada nó deve ter em sua instalação do *Hadoop* vários arquivos xml, que são responsáveis pela configuração das MVs do *Hadoop* naquela máquina.

Para conhecimento, esses arquivos são: *core-site.xml, yarn-site.xml, mapred-site.xml* e *hdfs-site.xml*. Cada um desses arquivos terá a configuração de um serviço do *Hadoop*. O arquivo *hdfs-site.xml* por exemplo é responsável pela configuração do HDFS naquela máquina. É importante salientar que esta configuração em nenhum momento é realizada automaticamente pelo *Hadoop* e que o usuário deve configurar cada nó separadamente.

2.1.3 *MapReduce*

O paradigma de *MapReduce*, já citado várias vezes no presente trabalho, divide o processamento em duas etapas. Essas duas etapas são derivadas das funções *Map* e *Reduce* das linguagens funcionais, e assim como nas funções oiginais, elas tem funcionamento baseado em tuplas de chave e valor. Um ciclo de aplicação típico é a função *Map* receber um arquivo de entrada e buscar os valores procurados pela aplicação, então formar tuplas de chave e valor. Feitas as tuplas, a função *Map* manda o resultado para a *Reduce* onde as chaves serão processadas e reduzidas a dados mais significativos. A grande vantagem do *Hadoop* é que dado um ambiente corretamente configurado, o programador pode focar sua atenção à resolução das tarefas pelo paradigma do *MapReduce* e não em como o trabalho será distribuído.

2.2 Sensibilidade ao contexto

Dada a interligação dos sistemas hoje em dia, já é possível notar alguma sensibilidade ao contexto na maioria deles. Ao acessar um site por um dispositivo móvel, o site automaticamente irá carregar sua versão *mobile*, a qual foi projetada para estes dispositivos, ou quando os navegadores utilizam dados de localidade para oferecer produtos, entre outros exemplos de utilização do contexto.

Segundo (**?**), sensibilidade ao contexto na computação se refere a habilidade de uma aplicação de detectar e responder as mudanças no ambiente de execução. O que leva a seguinte definição feita por (**?**), onde ele afirma que um sistema sensível ao contexto é capaz de adaptar suas operações ao contexto atual sem intervenção explicita do usuário e portanto aumentar sua usabilidade e eficácia.

Partindo dessas duas afirmações vem a dúvida sobre o que seria o contexto, portanto a definição do contexto é fundamental para que haja um entendimento da sensibilidade ao contexto (**?**). O contexto pode assumir diversos significados dependendo da situação que se encon-

tra, (**?**) define contexto como qualquer informação que pode ser utilizada para caracterizar a situação de uma entidade (pessoa, lugar ou objeto) considerado relevante para a interação entre usuário e aplicação.

Geralmente informações de contexto são utilizadas para a melhoria de performance de um sistema ou algoritmo, portanto estima-se ser possível melhorar a execução do *Apache Hadoop* através da utilização dessa técnica.

Embora existam diversas maneiras de se utilizar essas informações de contexto para melhorar a performance do *MapReduce*, (**?**) cita três exemplos de como isso pode ser feito, os quais se resumem em: configuração automática dos nós durante a instalação, gerenciamento de entrada e saída de nós do *cluster* e finalmente na distribuição de tarefas feita pelo escalonador de acordo com a disponibilidade de recursos e tarefas já em execução. A terceira maneira apresentada é a maneira que corresponde à abordagem utilizada nesse trabalho.

2.3 Escalonadores para Hadoop

Uma dos principais componentes do *Hadoop* é o escalonador, componente responsável pela distribuição do trabalho no ambiente. Além dos escalonadores disponibilizados juntamente com o próprio *Hadoop*, existem outras implementações que buscam solucionar uma necessidade específica que os escalonadores padrões não oferecem suporte.

2.3.1 *Hadoop Internal Scheduler*

O escalonador padrao do *Hadoop* foi implementado visando suportar apenas a submissão de tarefas em lote. Nesse escalonador, a primeira tarefa recebida é a primeira executada, formando uma fila para as subsequentes. Apesar de simples este escalonador também suporta cinco níveis de prioridade, porém a escolha da próxima tarefa nunca deixará o tempo de submissão completamente de fora.

2.3.2 *Fair Scheduler*

Utilizado para computar tarefas pequenas em lote que possuam os mesmos dados de entrada, utilizando um escalonamento em dois níveis para distribuir recursos igualitáriamente (**?**). O nível superior, geralmente aloca filas para cada usuário, utilizando um algoritmo justo com pesos. O segundo nível aloca os recursos dentro de cada fila, e utiliza um algoritmo igual

ao *Internal Scheduler*.

2.3.3 *Capacity Scheduler*

Este escalonador surgiu para os casos onde um ambiente *Hadoop* é dividido entre várias empresas ou possui partes distribuídas em diversos locais sob responsabilidade de mais de um dono. Ele é focado em garantias de que uma quantidade mínima de recursos será disponibilizada a qualquer momento que um de seus usuários decidir utilizar o *Hadoop*. O benefício decorre que organizações diferentes possuem picos de processamento em horas diferentes, portanto as organizações que estão utilizando o *Hadoop* irão se aproveitar da capacidade ociosa das outras.

2.4 Trabalhos relacionados

Foi feita uma pesquisa bibliográfica com objetivo de analisar os trabalhos que já haviam sido desenvolvidos envolvendo o *Hadoop* e que se propunham a alterar ou adaptar o escalonador. Além disso, buscou-se identificar quais técnicas eram as mais utilizadas e em cima de quais objetivos o trabalho foi desenvolvido. A seguir encontram-se os trabalhos relacionados e um breve resumo sobre a proposta, contexto utilizado e objetivo esperado com as alterações.

- CASH (*Context Aware Scheduler for Hadoop*) (**?**), nesse trabalho o objetivo dos autores é de melhorar o rendimento geral do *cluster*. Eles partem da hipótese de que grande parte dos jobs são periódicos e executados no mesmo horário, além de possuírem características de uso de CPU, rede, disco etc. semelhantes. O trabalho ainda leva em consideração que com o passar do tempo os nós tendem a ficar mais heterogêneos. Com a intenção de solucionar esses problemas e baseados nessas hipóteses, foi implementado um escalonador que classifica tanto os jobs como as máquinas com relação ao seu potencial de CPU e E/S, podendo então distribuir os jobs para máquinas que tem uma configuração apropriada para sua natureza.

- LATE (*Longest Approximation Time to End*) (**?**), seguindo o que o nome sugere, nesse tabalho a informação de contexto é referente ao tempo estimado de término da *task* baseado numa heurística que faz a relação de tempo decorrido e *score*. Essa informação é usada também para gerar um limiar de quando uma *task* é lenta o suficiente para indicar sitomas de erros e então iniciar uma nova em outra máquina possívelmente mais rápida. O

objetivo do trabalho era de reduzir o tempo de resposta em *clusters* grandes que executam muitos *jobs* de pequena duração.

- *A Dynamic MapReduce Scheduler for Heterogeneous Workloads* (**?**), aqui os autores também utilizam a técnica de classificar os *jobs* e máquinas de acordo com a quantidade de E/S ou CPU. E assim como no CASH, o principal objetivo é a melhora de rendimento no *cluster*. Uma das diferenças, no entanto, é que essa implementação utiliza um escalonador com três filas.

- SAMR (*A Self-adaptative MapReduce*) (**?**), essa implementação segue a mesma ideia do LATE, onde a informação de contexto é referente ao cálculo do progresso de uma *task* para identificar se é necessário lançar outra task igual ou não. Porém essa solução varia o cálculo do progresso de acordo com informações do ambiente em que a *task* está sendo executada, seu principal objetivo é a redução do tempo de execução das *tasks*. Para que sejam utilizadas informações do ambiente, o algoritmo leva em consideração informações históricas contidas em cada nó, e ajusta o peso de cada estágio do processamento.

- COSHH (*A Classification and Optimization based Scheduler for Heterogeneous Hadoop Systems*) (**?**), um pouco mais abrangente das demais soluções apresentadas, essa solução leva em consideração informações não especificadas sobre o sistema. Seu ganho de performance se dá a partir da classificação dos *jobs* em classes, ele então faz uma busca por máquinas que se encaixem nessa mesma classe. Essa busca é feita por um algoritmo que reduz o tamanho do espaço de busca para melhorar o rendimento. O objetivo dessa solução é a melhora do tempo médio em que os *jobs* são completados, além de oferecer uma boa performance quando utilizando somente o fatia mínima, além de proporcionar uma distribuição justa.

- *Quincy* (**?**), diferentemente de todos os outros trabalhos, essa solução não foi desenvolvida visando somente o *Hadoop* mas ainda assim é aplicável ao mesmo. Possuindo como objetivo melhorar o desempenho geral de um *cluster*, utiliza como informação de contexto a distribuição de recursos e modifica a maneira tradicional de tratamento desses. Ao utilizar um modo mais dinâmico do que o convencional, a solução mapeia os recursos num grafo de capacidades e demandas e calcula o escalonamento ótimo a partir de uma função global de custo.

- *Improving MapReduce Performance through Data Placement in heterogeneous Hadoop Clusters* (**?**), buscando melhorar a performance de *jobs* que possuam muito processamento de dados através da melhor distribuição desses dados, essa solução utiliza principalmente a localidade dos dados como informação para tomada de decisões. O ganho de performance é dado pelo rebalanceamento dos dados nos nós, deixando nós mais rápidos com mais dados. Isso diminui o custo de *jobs* especulativos e de transferência de dados pela rede.

Após estudo dos trabalhos, nota-se que muitos deles tem por objetivo a diminuição do tempo de resposta ou a melhoria do rendimento de maneira geral, os quais diferem dos objetivos do presente trabalho, que na verdade busca proporcionar uma melhor adaptação do *Hadoop* a um ambiente heterogêneo.

Constata-se também que há uma diversidade de contextos levados em consideração, contudo é possível identificar temas recorrentes como : a classificação dos *jobs* e dos nós quanto ao potencial de E/S ou de CPU, a avaliação do progresso da *task* na decisão de lançar ou não uma nova *task* especulativa.

3 DESENVOLVIMENTO

Este capítulo descreve as etapas realizadas para atingir os objetivos desse trabalho. O trabalho foi dividido em quatro etapas, dada a complexidade do *framework* em questão, essa decisão foi tomada visando primeiramente entender o contexto ao qual o escalonador terá de se adaptar e de que forma isso será feito, deixando a etapa de implementação para o final quando todos os pré-requisitos para a inserção no *Hadoop* já estiverem concluídos. A primeira etapa visou à instalação e configuração das versões 1.0.4 e 2.0.3 (YARN), para aprofundar o conhecimento sobre os requisitos do ambiente para que o *framework* seja utilizado. Na segunda etapa foram feitas as instalações dos componentes necessários para a compilação do código, visto que o objetivo do trabalho é alterar o código fonte do Apache Hadoop. A terceira etapa foi destinada ao estudo da arquitetura do *Apache Hadoop*, através do código dos escalonadores disponibilizados e da máquina de estados do *Resource Manager*. Finalmente a quarta etapa é a de desenvolvimento do escalonador, o qual será posteriormente testado tanto em ambiente local como no Grid'5000.

3.1 Configuração do ambiente de execução

A instalação do *Apache Hadoop*, ao contrário da instalação dos programas que usuários finais estão habituados, não possui interface gráfica. Na verdade a instalação não passa da extração dos arquivos para uma pasta, porém a configuração do ambiente não é trivial e exige um conhecimento de administração de sistemas e redes.

Para o correto funcionamento do *Apache Hadoop*, é necessária a edição de alguns arquivos que descrevem o ambiente no qual esse irá rodar. Além disso, é necessária a configuração de uma rede onde os nós possam se comunicar livremente. Sabendo que o *Apache Hadoop* sofreu algumas alterações ao mudar da versão 1.x para o 2.x (YARN), esperava-se que ao realizar a instalação e configuração das duas versões, a identificação das diferenças entre elas torna-se mais simples.

No início do trabalho o objetivo era de instalar e configurar o *Apache Hadoop* em três situações:

- *localhost*, também chamado de *single-node*
- *mini cluster*, chamado de *multi-node*

- Grid'5000, sendo também um caso de *multi-node*

São duas as etapas de configuração, a configuração da rede e comunicação a qual é referente à configuração independente do hadoop, ou seja, são tarefas a serem realizadas dentro do sistema operacional, tal como a criação de usuário e configuração do ssh. Enquanto isso, a configuração do ambiente é referente à configuração específica do *Apache Hadoop*, portanto têm de alterar arquivos de configuração dentro pasta de instalação desse de tal forma que ao serem executados, os deamons do *Apache Hadoop* rodem sem problemas.

A configuração do sistema operacional é exatamente igual em ambas versões, desde a criação de um usuário exclusivo para o *Apache Hadoop* até a criação das chaves públicas de SSH.

Quanto à configuração do ambiente existem algumas diferenças entre as versões, mas ambas seguem a mesma forma geral de configuração, com pequenas diferenças pontuais em alguns casos. Essa configuração é feita através da edição de arquivos xml, nos quais são inseridos propriedades com nome e valor, as quais irão alterar o funcionamento ou a classe específica que será utilizada na execução. A maior diferença entre as versões é a adição de um novo arquivo na versão YARN, que possui muitos parâmetros importantes, por se tratar da configuração do módulo do *MapReduce*.

Inicialmente optou-se por configurar ambas as versões em todas as instâncias especificadas, porém devido ao foco do projeto a utilização da versão 1.x não seria mais interessante e foi descontinada.

A configuração no ambiente *localhost*, é simples e ocorre sem problemas após um contato básico com o *framework*. A partir desse ponto, a complexidade começa a aumentar, visto que são inseridos mais nós e existe a necessidade de um gerenciamento melhor, além da inserção de mais propriedades nos arquivos.

Para a versão *mini cluster*, foram utilizadas duas máquinas, já apresentando uma diferença entre as versões. Na versão 1.x o mestre poderia também ser escravo, ou seja, uma máquina poderia ser *NameNode, DataNode, TaskManager e TaskTracker*. Na versão YARN isso já muda e a máquina de gerenciamento não pode executar *deamons* de processamento.

Para a configuração no Grid'5000, foi decidido utilizar somente a versão YARN. Uma vez que a versão *mini cluster* já tinha sido configurada, foram necessárias poucas adaptações. A principal diferença foi quanto à configuração interna do Grid'5000, tendo que ser aplicadas ferramentas disponibilizadas para utilização e criação de imagens de sistema, além de algumas

diferenças para obter os *tarballs* necessários para os testes de instalação e execução, que só poderiam ser obtidos com wget através de um *proxy* ou que se fosse alguma versão diferente da oficial, através de *scp*.

Ao final desta etapa, alguns pontos do *Hadoop* já tinham sido esclarecidos e também já havia sido estabelecido um ambiente de execução onde os testes serão feitos ao final da implementação.

Ainda como resultado desta etapa, destaca-se a identificação de algumas peculiaridades do *framework* que poderiam dificultar o trabalho, como o caso de inserção de um novo nó escravo. Para que esse nó seja inserido, os serviços tem de ser parados, para que o usuário altere o arquivo de escravos e então reinicie os serviços. O mesmo acontece para qualquer propriedade que deseja-se alterar nos arquivos *xml*, não há possibilidade de alterá-las sem que os serviços sejam interrompidos e reiniciados.

3.2 Alteração e compilação do código fonte

Dada a natureza do projeto, de gerar um novo escalonador sensível ao contexto para o *Apache Hadoop*, é necessário que esse escalonador seja incluído na distribuição final e que fique disponível para o uso de todos usuários que fizerem o *download* da versão alterada.

Para que isso seja alcançado, é necessário gerar novos arquivos *.jar* a partir do código fonte modificado, ou nesse caso, adicionado. Em razão disto foi feito um estudo prévio das etapas e requisitos necessários para a compilação e geração desses arquivos.

Primeiramente foi feita uma consulta à documentação oficial e aos arquivos de suporte inclusos nas distribuições do código fonte. Adquirindo-se assim um guia do que era necessário e quais as etapas a seguir.

Partindo da lista de requisitos nas fontes de informação, estes foram instalados. Alguns foram facilmente instalados pelo próprio gerenciador de pacotes do GNU outros requeriram um esforço maior, mas no geral esta etapa decorreu sem maiores problemas.

A última etapa era a compilação própriamente dita, onde bastava executar o comando do *Apache Maven* (**?**) e esperar enquanto ele fazia o download de requisitos e a construção dos arquivos executáveis. Existem alguns parâmetros que podem ser adicionados ou retirados no momento da compilação de acordo com a intenção do usuário, como por exemplo a opção de pular os testes, a opção de compilar código nativo e documentação, entre outras.

Houveram problemas na etapa de compilação própriamente dita, onde a ferramenta de

compilação *Apache Maven* causava alguns erros em função dos arquivos de configuração utilizados por ela (arquivos *pom.xml*). Através de uma busca nos fóruns de desenvolvedor da comunidade *Hadoop*, foi possível identificar que o erro era referente à versão de um dos requisitos (*Protocol Buffer* (**?**)), o qual encontrava-se com uma versão desatualizada no arquivo de configuração, bastando alterar a versão ou aplicar o patch de correção disponibilizado pela equipe para que o erro fosse sanado.

Embora a compilação do código padrão tenha sido bem sucedida, o maior objetivo deste processo era descobrir como a compilação decorreria com a adição de novas classes no código padrão. Com vistas a atingir esse objetivo, uma nova classe de escalonador foi adicionada. Essencialmente essa classe era apenas uma cópia do escalonador FIFO renomeada e em outro *package*. Após gerar os arquivos executáveis com a adição dessa nova classe, esses arquivos foram copiados para o Grid'5000 e foi feita uma nova instalação.

Com essa nova instalação em mãos, foi possível testar o funcionamento e compará-lo com o da distribuição padrão. Além disso foram feitos testes para que fosse comprovado a utilização do "novo"escalonador, os quais foram concluídos com sucesso, mediante uma linha extra de código identificando que o escalonador utilizado era essa versão alterada. Seguindo na mesma vulnerabilidade identificada na instalação do ambiente de execução, a opção de qual escalonador será utilizado é mediante arquivo *xml* e também deve ser feita antes do serviço *ResourceManager* ser iniciado, caso contrário esse serviço terá de ser parado e reiniciado.

3.3 Estudo da arquitetura do *Apache Hadoop*

Com intuito de criar um escalonador sensível ao contexto é necessário que a arquitetura do *Apache Hadoop* seja compreendida. Esta etapa do projeto tem objetivos de identificar as dependências entre as classes, além de quais classes serão necessárias implementar e como se dará essa implementação.

Uma vez que um escalonador funcional é uma implementação de uma série de classes abstratas e *interfaces*, é um bom começo saber a origem de tais classes e o que cada uma irá influenciar na classe final. Ainda, através do estudo da arquitetura, é possível identificar os recursos disponibilizados pelo *framework* para que então sejam decididas as estratégias de escalonamento.

Para o estudo da arquitetura foram utilizados dois métodos. O primeiro método empregado consistiu no estudo do próprio código enquanto o segundo pela geração e estudo de

máquinas de estado do *Resource Manager*. Esse passo foi destinado à identificação de todos os componentes de um escalonador, desde a identificação de quais *interfaces* e classes abstratas foram implementadas até o estudo e investigação dos critérios de escalonamento e como estes são aplicados nos escalonadores disponibilizados.

Devido à complexidade do *framework* decidiu-se por utilizar a *IDE IntelliJIDEA* (**?**) na execução do primeiro método. Terminado o estudo do código, foi possível gerar um diagrama de classes daquelas que compõem um escalonador, o qual é apresentado na Figura **??**.

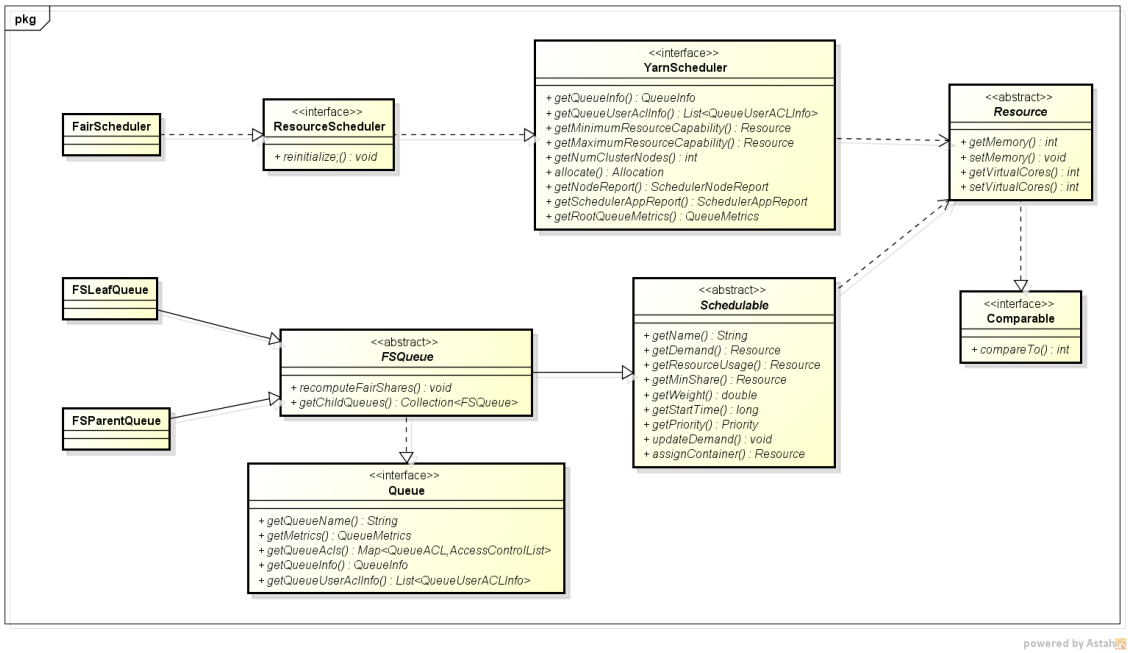

Figura 3.1 – Diagrama de classe com as principais classes que compõem o *FairScheduler*

Identificou-se também a ideia base de cada componente, as quais serão explicadas a seguir:

- *Schedulable*: uma classe abstrata que representa uma entidade que pode lançar um *job* ou uma *queue*, disponibiliza uma interface simples pela qual os algoritmos podem ser aplicados tanto dentro de uma *queue* quanto a várias delas.
- *Queue*: uma *interface* que possibilita o controle básico de todas as *queues* de um escalonador.
- *YarnScheduler*: outra *interface* através da qual os componentes podem se comunicar com o escalonador, seja para alocar ou liberar recursos.

- *Resource*: uma classe abstrata responsável por modelar os recursos de um computador (memória, cores etc.) a serem utilizados.

- *Comparable*: uma *interface* que possibilita comparar vários tipos de dados, utilizado para comparar os *jobs* submetidos e ver qual tem prioridade segundo os critérios de escalonamento adotados.

O segundo método foi realizado mediante uma opção disponibilizada pelo próprio *Maven*, em que é possível gerar grafos que representam as máquinas de estado referentes ao funcionamento do *Resource Manager* e do *Node Manager*. Este método apresentou como resultado os grafos referentes às máquinas de estados geradas a partir do *Resource Manager*, é possível identificar que são máquinas de estado de sub-componentes dele. Ainda constata-se que as figuras podem ajudar a identificar o fluxo de um *job*, o que será importante na etapa de testes em situações adversas. As figuras podem ser visualizadas no **??** do trabalho.

Terminada essa etapa, todos os requisitos necessários para a implementação já estão concluídos, já existe um ambiente de execução para os testes, já existe um ambiente de compilação e implementação do código, os recursos possíveis de serem utilizados já estão identificados e as alterações de configuração necessárias na instalação do *Hadoop* para que um novo escalonador seja utilizado são conhecidas.

4 PRÓXIMAS ETAPAS

As próximas etapas deste projeto são as seguintes:

1. Decisão das métricas do algoritmo de escalonamento com base nos recursos identificados e suportados pelo *Hadoop*
2. Implementação do escalonador
3. Testes e avaliação
4. Ajustes e novos testes se necessário

APÊNDICES

APÊNDICE A – Grafos gerados referentes ao *ResourceManager*

A seguir encontram-se uma breve explicação e as figuras geradas pelo segundo método empregado na etapa de estudo da arquitetura do *framework*.

A pertinência das figuras é validada por elas descreverem todos os caminhos possíveis que dada interface pode tomar. O fluxo do caso perfeito seria iniciado com o recebimento de um novo *job* pelo *ResourceManager*. Para que o *job* possa ser iniciado alguns pré-requisitos devem ser satisfeitos. É a partir desse ponto que as figuras entram em cena.

Para que uma aplicação seja iniciada, primeiramente é criado um *AppAttempt*. O *AppAttempt* é literalmente uma tentativa de aplicação, pela qual o *ResourceManager* tentará alocar os recursos necessários (*Node e Container*) para que essa aplicação seja realmente criada e submetida ao cluster. Durante o processo do *RMAppAttempt* o *ResourceManager* irá tentar alocar um ou mais *RMContainers* bem como *RMNodes* para que a aplicação tenha recursos suficientes para sua execução. Após os recursos necessários terem sido alocados com sucesso, a aplicação passa para o status de *RMApp* e é submetida ao cluster.

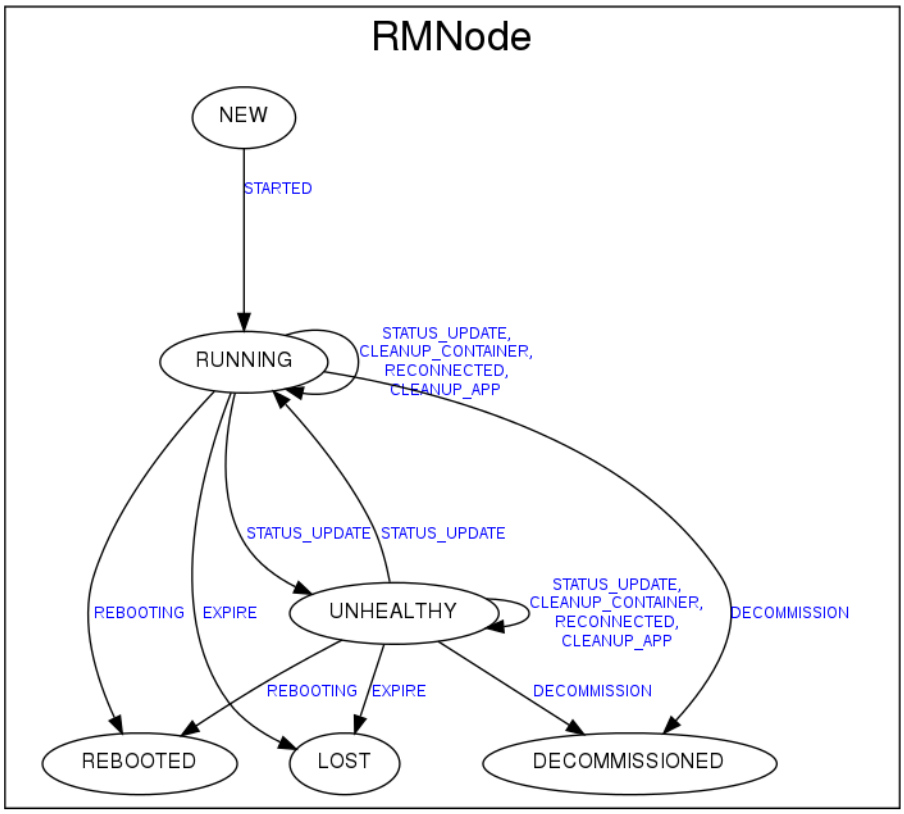

Figura A.1 – Máquina de estados do RMNode

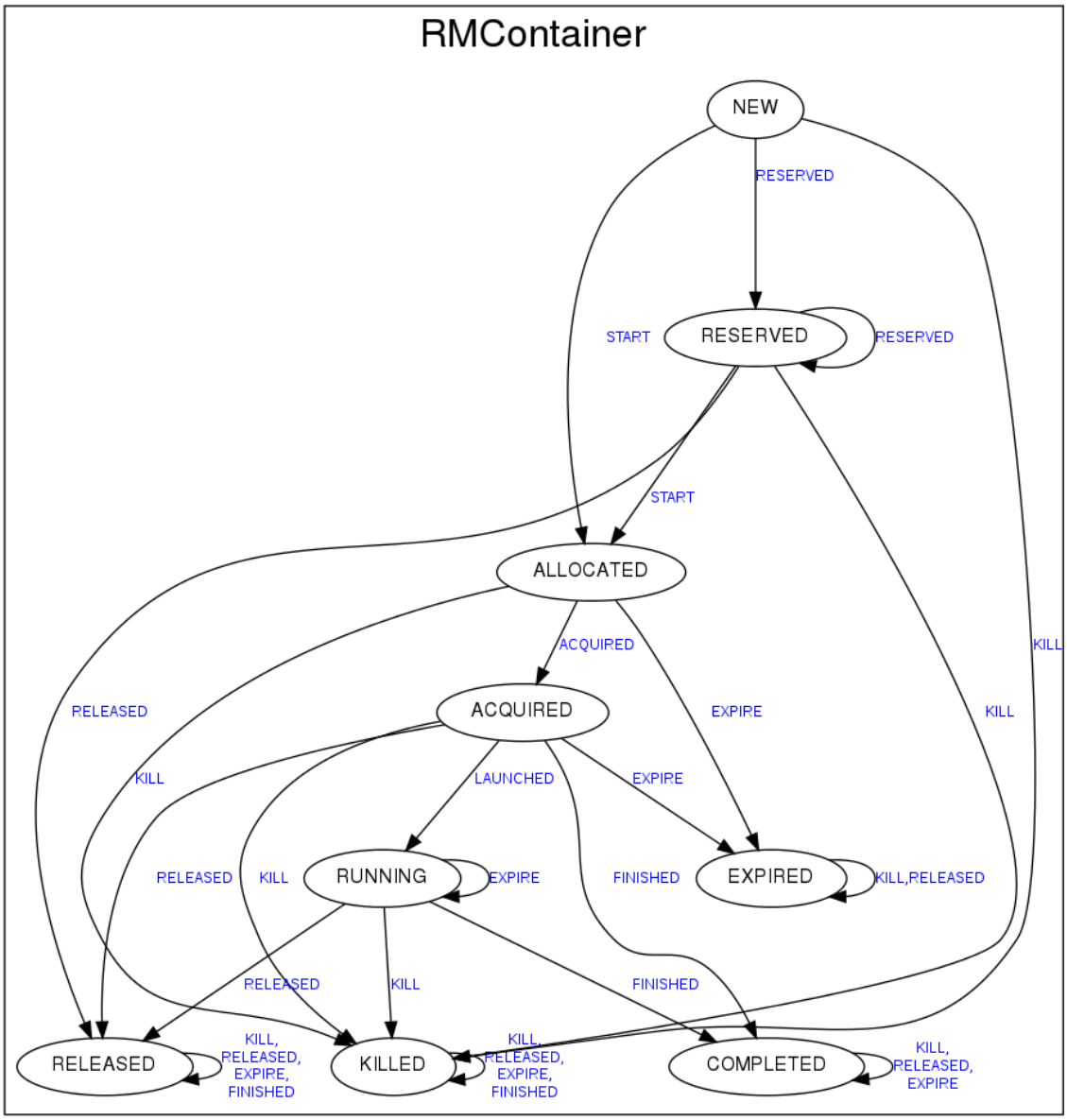

Figura A.2 – Máquina de estados do RMContainer

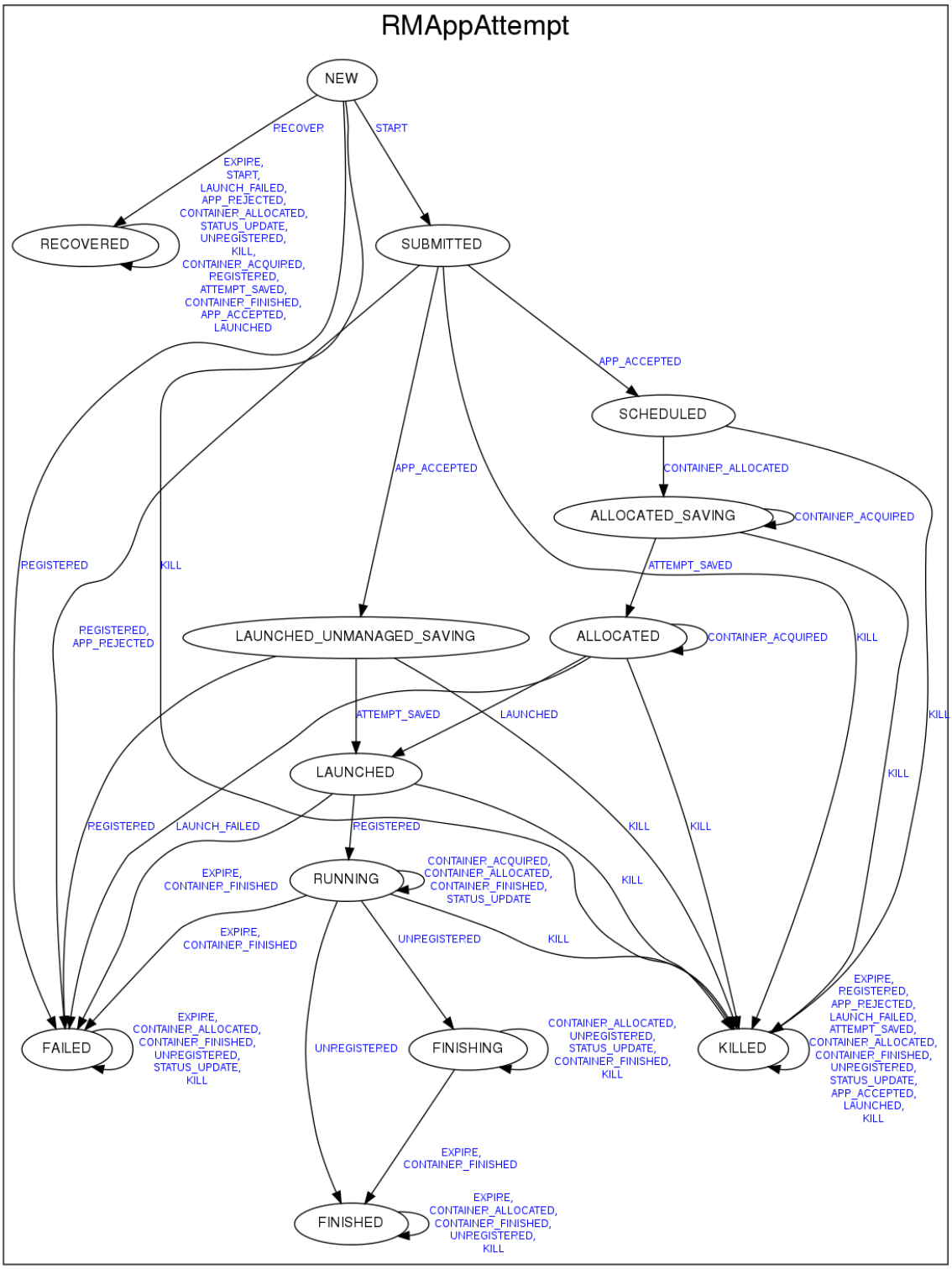

Figura A.3 – Máquina de estados do RMAppAttempt

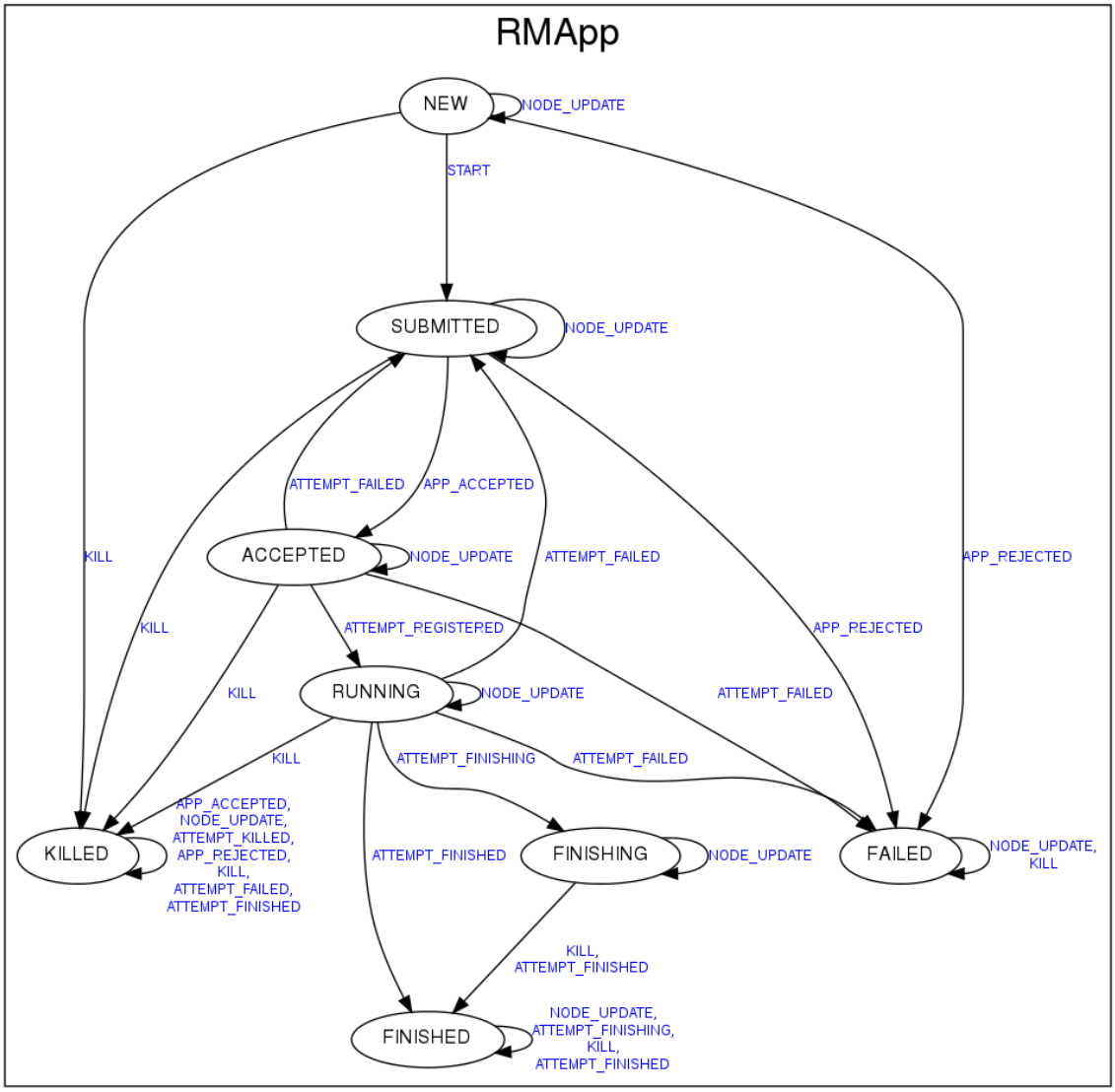

Figura A.4 – Máquina de estados do RMApp