UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS MOURA DANTAS

UMA ESTRATÉGIA PARALELA E DISTRIBUÍDA PARA ASSEGURAR A CONFIDENCIALIDADE DE DADOS ARMAZENADOS EM NUVEM

FORTALEZA

2017

LUCAS MOURA DANTAS

UMA ESTRATÉGIA PARALELA E DISTRIBUÍDA PARA ASSEGURAR A CONFIDENCIALIDADE DE DADOS ARMAZENADOS EM NUVEM

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciências da Computação. Área de Concentração: Banco de Dados

Orientador: Prof. Dr. José Maria da Silva Monteiro Filho

Coorientador: Prof. Dr. João Paulo do Vale Madeiro

FORTALEZA

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

D213e Dantas, Lucas.
Uma estratégia paralela e distribuída para assegurar a confidencialidade de dados armazenados em nuvem / Lucas Dantas. – 2017.
86 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2017.
Orientação: Prof. Dr. José Maria da Silva Monteiro Filho.
Coorientação: Prof. Dr. João Paulo do Vale Madeiro.

1. Privacidade. 2. Confidencialidade. 3. Computação em Nuvem. I. Título.

CDD 005

LUCAS MOURA DANTAS

UMA ESTRATÉGIA PARALELA E DISTRIBUÍDA PARA ASSEGURAR A CONFIDENCIALIDADE DE DADOS ARMAZENADOS EM NUVEM

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciências da Computação. Área de Concentração: Banco de Dados

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. José Maria da Silva Monteiro Filho (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. João Paulo do Vale Madeiro (Coorientador)
Universidade da Integração Internacional da Lusofonia Afro-Brasileira (UNILAB)

Prof. Dr. Javam de Castro Machado
Universidade Federal do Ceará (UFC)

Prof. Dr. Angelo Roncalli Alencar Brayner
Universidade Federal do Ceará (UFC)

Prof. Dr. Flávio Rubens de Carvalho Sousa
Universidade Federal do Ceará (UFC)

Prof. Dr. José de Aguiar Moraes Filho
Universidade de Fortaleza (UNIFOR)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe e Pai, vocês foram essenciais para alcançar meus objetivos. Obrigado meu Deus por tudo!

AGRADECIMENTOS

Aos meus pais Hélio e Maria, pela presença, dedicação, exemplo, amor e compreensão. Sou muito grato por tudo que são e que aspiram ser. Espero um dia ser tão especial para alguém como são para mim.

Aos meus irmãos, Matheus e Gabriel, pelos exemplos de perseverança, ajuda nos momentos difíceis, amizade e convivência.

Aos meus familiares, pela presença, convivência, amizade, troca de conhecimentos e pelo amor. Em especial minha madrinha Sofia, pelos conselhos e conversas de longas horas.

Ao meu orientador, Prof. Dr. José Maria da Silva Monteiro Filho. Me sinto abençoado por receber alguém tão compromissado, dedicado, compreensivo, capacitado e amigo para ser meu orientador.

Ao meu coorientador, Prof. Dr. João Paulo Madeiro, pela presença, apoio, cobrança e dedicação.

Aos colegas do ARIDA, pelos conselhos, auxílios e pela amizade.

Aos professores do Mestrado e Doutorado em Ciências da Computação - MDCC, pelos valiosos ensinamentos.

Aos colegas do MDCC, que pelo convívio e troca de conhecimentos se tornaram amigos para toda a vida.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

A todos aqueles que, direta ou indiretamente, contribuíram para a realização deste trabalho.

“ You want to know the great thing about today?
It doesn’t care about what you did or didn’t do
yesterday or what you’re going to do tomorrow.
It only cares about what you do right now.”

(Noel DeJesus)

RESUMO

Atualmente, o armazenamento de grandes quantidades de dados confidenciais em servidores na nuvem é uma prática comum, uma vez que permite reduzir custos e aumentar a disponibilidade desses dados. Porém, nos ambientes de computação em nuvem, o controle dos dados deixa de ser do seu proprietário e passa a ser do provedor do serviço de armazenamento, o que faz surgir novos desafios relacionados à privacidade, segurança e confidencialidade. Neste contexto, diferentes soluções para assegurar a confidencialidade dos dados armazenados na nuvem foram propostas. Em geral, tais estratégias utilizam criptografia, fragmentação de dados ou uma combinação dessas duas abordagens. Recentemente, uma nova solução, denominada QSM-EXTRACTION, foi proposta. A estratégia QSM-EXTRACTION baseia-se na fragmentação de um arquivo digital em fragmentos denominados objetos de informação, na decomposição desses objetos por meio da extração de suas características e na dispersão dessas características em diferentes serviços de armazenamento em nuvem. Contudo, apesar de ter sido concebida para o ambiente de computação em nuvem, a QSM-EXTRACTION adota uma abordagem de execução centralizada, o que pode comprometer o desempenho da etapa de decomposição. Neste trabalho, propomos uma versão paralela e distribuída da estratégia QSM-EXTRACTION, denominada pdQSM-EXTRACTION, a qual explora o paradigma MapReduce com a finalidade de possibilitar uma maior eficiência no processo de extração das características dos objetos de informação. A abordagem pdQSM-EXTRACTION foi implementada em linguagem Scala utilizando-se *o framework Apache Spark*. Diversos experimentos foram realizados para avaliar a estratégia proposta. Os resultados obtidos demonstram que para arquivos com tamanhos maiores ou iguais a 4GB, a estratégia pdQSM-EXTRACTION apresenta desempenho melhor que o obtido pela estratégia QSM-EXTRACTION, avaliado pelo cálculo do tempo de entrada, definido como o tempo total gasto para decompor um dado arquivo gerando-se outros três arquivos contendo as características de qualidade, quantidade e medida. Assim, considerando-se o processamento de arquivos com tamanhos maiores ou iguais a 4GB e a adição de um ou mais nós escravos pela estratégia pdQSM-EXTRACTION, a razão entre o tempo de entrada obtido pela estratégia pdQSM-EXTRACTION e o tempo de entrada obtido pela estratégia QSM-EXTRACTION apresentou valores mínimos e máximos respectivamente de 53,57% e 95,83%. Portanto, comprova-se viabilidade da utilização da abordagem pdQSM-EXTRACTION em aplicações envolvendo grandes volumes de dados.

Palavras-chave: Privacidade. Confidencialidade. Computação em Nuvem.

ABSTRACT

Nowadays, the storage of large amounts of confidential data in cloud servers is a common practice, since such strategy allows to reduce costs and also increases data availability. However, in cloud computing environment, data control is no longer owned by its legitimated user, becoming a storage service provider responsability. Such scenario gives rise to new challenges related to privacy, security and confidentiality. At this context, different solutions have been proposed for ensuring the confidentiality of the cloud stored data. In generall, such approaches are based on cryptography, data fragmentation or a combination of these two methodologies. Recently, a new approach, denoted QSM-EXTRACTION, has been proposed. The QSM-EXTRACTION strategy is based on the fragmentation of a digital file into fragments named information objects, on the decomposition of these objects through the extraction of some features and on the dispersion of these features in different cloud storage services. However, despite being developed for cloud computing environment, QSM-EXTRACTION method adopts a centralized execution approach, which may compromise the performance of the decomposition step. At the present work, we propose a paralell and distributed version of the QSM-EXTRACTION strategy, named pdQSM-EXTRACTION, which exploits the MapReduce paradigm aiming to provide a higher efficiency for the process of extracting features from information objects. The pdQSM-EXTRACTION approach has been implemented in Scala language programming, using Apache Spark framework. Several computing experiments and simulations have been performed aiming to evaluate the proposed approach. The obtained results, considering file sizes greater than or equal to 4GB, show that pdQSM-EXTRACTION strategy presents better performance than the one obtained by the QSM-EXTRACTION strategy, evaluated by computing the input time, defined as the total time spent to decompose a given file generating three other files containing the characteristics of quality, quantity and measurement. Thus, considering the processing of files whose sizes are greater than or equal to 4GB and the addition of one or more slave nodes by the pdQSM-EXTRACTION strategy, the ratio between the input time obtained by the pdQSM-EXTRACTION strategy and the input time obtained by the QSM-EXTRACTION strategy presented minimum and maximum values respectively of 53.57 % and 95.83 %. Therefore, we achieve to demonstrate the feasibility of pdQSM-EXTRACTION approach for applications involving large data volumes.

Keywords: Privacy. Confidentiality. Cloud Computing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AICPA	American Institute of Certified Public Accountants
ASCII	American Standard Code for Information Interchange
CICA	Canadian Institute of Charted Accountants
IaaS	Infrastructure as a Service
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
PII	Personally Identifiable Information
SaaS	Software as a Service

SUMÁRIO

1 INTRODUÇÃO

Este trabalho apresenta uma versão paralela e distribuída da estratégia QSM-EXTRACTION (BRANCO *et al.*, 2016c), denominada pdQSM-EXTRACTION. A estratégia QSM-EXTRACTION baseia-se na fragmentação de um arquivo digital em fragmentos denominados objetos de informação, na decomposição desses objetos por meio da extração de suas características e na dispersão dessas características em diferentes serviços de armazenamento em nuvem, permitindo a posterior recuperação desses dados sem perda de informação. Desta forma, a QSM-EXTRACTION busca inviabilizar a reconstrução do arquivo original por parte de um provedor de nuvem que possui apenas parte das características dos objetos de informações que compõem um determinado arquivo. Assim, assegura-se a confidencialidade dos dados armazenados em nuvem e, por conseguinte, a privacidade dos proprietários desses dados. Todavia, apesar de ter sido concebida para o ambiente de computação em nuvem, a estratégia QSM-EXTRACTION adota uma abordagem de execução centralizada, o que pode comprometer o desempenho da etapa de decomposição. Com a finalidade de superar esta limitação, investigamos a utilização do Paradigma MapReduce na etapa de extração das características dos objetos de informação, o que deu origem à abordagem pdQSM-EXTRACTION (implementada em linguagem Scala utilizando-se o *framework Apache Spark*). Diversos experimentos foram realizados para avaliar a estratégia proposta. Os resultados obtidos comprovam a viabilidade da utilização da abordagem pdQSM-EXTRACTION em aplicações que envolvem grandes volumes de dados (arquivos com tamanho acima de 1 *Gigabyte*).

1.1 Motivação

A revolução proporcionada pela tecnologia da informação, principalmente com o surgimento das redes sociais e a massificação dos dispositivos portáteis, intensificou a geração e divulgação de informações pessoais. Atualmente, uma variada gama de aplicativos e dispositivos emitem, constantemente, informações sobre localização, relações sociais, hábitos de consumo, comportamento, etc. Segundo Keen et. al. (KEEN, 2012) “Informações pessoais são o novo petróleo, o combustível vital da nossa economia digital”. Para as organizações, dados pessoais sobre os clientes e potenciais clientes são agora um elemento essencial. A capacidade de monitoramento detalhado das ações das pessoas na *Web* e a prática difundida das empresas compartilharem e venderem dados umas para as outras afetam a privacidade de quem acessa a

Web.

Adicionalmente, o desenvolvimento da computação em nuvem proporcionou o surgimento de serviços de armazenamento de dados, tais como Dropbox, Google Drive, Amazon Cloud Drive, Box, iCloud, OneDrive, dentre outros. Atualmente, muitos usuários utilizam esses serviços para armazenar arquivos pessoais, tais como, fotos, documentos, além de arquivos de áudio e vídeo. Nestes casos, o controle dos dados armazenados na nuvem deixa de ser do proprietário do dado e passa a ser do provedor do serviço de armazenamento.

Por estes motivos, o debate sobre a privacidade no mundo moderno tem obtido cada vez mais destaque. Certamente, os usuários da *Web* não desejam que suas informações pessoais estejam acessíveis a pessoas não autorizadas. Uma pesquisa realizada em 2015 pela empresa Unisys (*Unisys Security Insights 2015: Consumer viewpoints on security*) revelou que mais da metade dos brasileiros entrevistados (cerca de 53%) têm medo que seus dados pessoais sejam violados (LIEBERMAN, 2015).

Contudo, ainda existe muito debate acerca dos termos privacidade, segurança e confidencialidade. O *American Institute of Certified Public Accountants* (American Institute of Certified Public Accountants (AICPA)) e o *Canadian Institute of Charted Accountants* (Canadian Institute of Charted Accountants (CICA)) definem que, “Privacidade é o conjunto de direitos e obrigações que indivíduos e organizações têm em relação à coleta, uso, retenção e divulgação de informações pessoais". O *National Institute of Standards and Technology* (National Institute of Standards and Technology (NIST)) define segurança computacional como sendo “a proteção conferida a um sistema de informação automatizado, a fim de atingir os objetivos propostos de preservação da integridade, disponibilidade e confidencialidade dos recursos do sistema de informação (incluindo *hardware*, *software*, *firmware*, informações/dados e telecomunicações)". Esta definição contém 3 conceitos chaves para segurança computacional: confidencialidade, disponibilidade e integridade (GUTTMAN; ROBACK, 1995).

A disponibilidade assegura que os dados estejam sempre disponíveis para o uso legítimo. A integridade está relacionada à garantia de que os dados manipulados mantenham todas as características originais estabelecidas pelo seu proprietário. Já a confidencialidade garante que os dados sejam acessados somente por entidades autorizadas pelo seu proprietário. Desta forma, a tríade CIA (*Confidentiality, Integrity and Availability*) – Confidencialidade, Integridade e Disponibilidade – orienta a análise, o planejamento e a implementação da segurança para uma determinada informação que se deseja proteger.

Vale enfatizar que a confidencialidade limita o acesso aos dados, impedindo a divulgação de dados não autorizados de forma intencional ou não intencional. Por outro lado, segundo Stallings et. al. (STALLINGS, 2010), a privacidade dos dados assegura que os indivíduos controlam ou influenciam quais informações relacionadas a eles podem ser coletadas e armazenadas por alguém e com quem elas podem ser compartilhadas. Portanto, confidencialidade e privacidade de dados são conceitos que mantêm uma íntima relação. Ambos relacionam-se com a necessidade de assegurar que o indivíduo controle que informações relacionadas a ele podem ser divulgadas e quem pode acessá-las. Desta forma, pode-se entender a confidencialidade como uma ferramenta para se obter privacidade.

Para tratar o problema de assegurar a confidencialidade de dados, várias abordagens foram propostas. Estas abordagens podem ser classificadas em quatro categorias (SAMARATI, 2014):

- uso de criptografia antes de enviar os dados para o servidor da nuvem;
- fragmentação vertical de dados e dispersão entre vários servidores na nuvem;
- combinação de fragmentação vertical e criptografia;
- extração das características dos dados e a dispersão dessas informações entre vários servidores na nuvem.

Existem questões abertas nas três abordagens previamente propostas. Em relação ao uso da criptografia, há uma troca entre a segurança dos dados e o desempenho das consultas (KANTARCIOĞLU; CLIFTON, 2005). Para se utilizar a fragmentação vertical é necessário, inicialmente, definir os fragmentos, separando em fragmentos diferentes os atributos com associação sensível, o que é um problema NP-difícil (SAMARATI; VIMERCATI, 2010; JOSEPH *et al.*, 2013). Já nas soluções mistas, que utilizam uma combinação da fragmentação vertical com a criptografia, somente os atributos sensíveis são encriptados e os atributos que possuem associação sensível são dispersos em fragmentos diferentes (FUGKEAW, 2012). Isto permite a execução de consultas sobre atributos não criptografados, mas ainda exige a decriptação dos atributos sensíveis. Além disto, esta abordagem continua com a necessidade de se determinar os fragmentos, o quê, como mencionado anteriormente, é um problema da categoria não polinomial difícil (NP-difícil) (pode ser reduzido ao problema de coloração do hipergrafo) (AGGARWAL, 2005). A decomposição dos objetos de informação por meio da extração de suas características foi proposta em (BRANCO *et al.*, 2014; BRANCO *et al.*, 2016a; BRANCO *et al.*, 2016b). Esta estratégia, denominada QSM-EXTRACTION, não utiliza chaves criptográficas e nem fragmen-

tação vertical. Porém, à medida que o tamanho dos arquivos digitais aumenta o desempenho do processo de decomposição dos objetos de informação diminui.

1.2 Definição do Problema

Apesar da estratégia QSM-EXTRACTION ter sido concebida para o ambiente de computação em nuvem, ela adota uma abordagem de execução centralizada. Este fato pode comprometer o desempenho do processo de decomposição dos objetos de informação por meio da extração se suas características, principalmente quando arquivos digitais contendo grandes volumes de dados são utilizados.

A hipótese desta dissertação é a seguinte: **O paradigma MapReduce pode ser utilizado para melhorar o desempenho da etapa de extração das características de informação, a qual compõe a estratégia QSM-EXTRACTION.**

1.3 Objetivo

O objetivo geral deste trabalho consiste em conceber, implementar e avaliar uma versão paralela da estratégia QSM-EXTRACTION. Mais precisamente, iremos explorar o paradigma MapReduce com a finalidade de possibilitar uma maior eficiência no processo de extração das características dos objetos de informação.

Para atingir este objetivo geral, foram definidos os seguintes objetivos específicos:

- Investigar o paradigma MapReduce;
- Investigar o *framework Apache Spark*;
- Projetar e implementar uma versão paralela e distribuída da estratégia QSM-EXTRACTION utilizando-se o paradigma MapReduce;
- Avaliar o desempenho da versão proposta por meio da realização de experimentos.

1.4 Contribuições

As principais contribuições desta dissertação são:

1. Uma nova estratégia paralela e distribuída, denominada pdQSM-EXTRACTION, para assegurar a confidencialidade de dados em serviços de armazenamento em nuvem;
2. Utilização do paradigma MapReduce;
3. Implementação da estratégia pdQSM-EXTRACTION, utilizando o ;

4. Avaliação da estratégia pdQSM-EXTRACTION;

1.5 Tipo de Pesquisa

Esta pesquisa pode ser classificada como quantitativa em relação à abordagem, uma vez que busca, por meio da objetividade e da utilização de variáveis mensuráveis, identificar em quais situações um determinado algoritmo é mais adequado que outro. Quanto à natureza, esta pesquisa pode ser classificada como aplicada, pois gera conhecimentos para a aplicação prática e soluções para problemas específicos. Em relação aos objetivos, esta é uma pesquisa exploratória, uma vez que foi realizado um levantamento bibliográfico, implementação e experimentos para maior familiarização com os problemas investigados. Quanto ao procedimento, esta é uma pesquisa experimental, pois, a partir das hipóteses levantadas, foram realizados diversos experimentos com diferentes variáveis com a finalidade de identificar o comportamento dos algoritmos.

1.6 Estrutura da Dissertação

Este estudo está estruturado em seis capítulos, além dessa introdução. O Capítulo 2 apresenta os conceitos fundamentais sobre privacidade, segurança de dados, computação em nuvem, teoria da informação e o *framework Spark*. Os trabalhos relacionados são discutidos no Capítulo 3. O Capítulo 4 apresenta uma nova estratégia paralela e distribuída, denominada pdQSM-EXTRACTION, para assegurar a confidencialidade de dados em serviços de armazenamento em nuvem. Os experimentos realizados com a finalidade de avaliar o desempenho da estratégia pdQSM-EXTRACTION são apresentados e discutidos no Capítulo 5. Por fim, o Capítulo 6 apresenta as conclusões desta tese e aponta direções para trabalhos futuros.

2 FUNDAMENTOS TEÓRICOS

2.1 Introdução

A computação em nuvem (*Cloud Computing*) é uma tecnologia que tem como objetivo proporcionar serviços de Tecnologia da Informação (TI) sob demanda com pagamento baseado no uso. A nuvem computacional é um modelo de computação em que dados, arquivos e aplicações residem em servidores físicos ou virtuais, os quais podem ser acessados por meio de uma rede, a partir de qualquer dispositivo compatível (fixo ou móvel), a qualquer hora, de qualquer lugar, sem a necessidade de instalação ou configuração de programas específicos (SOUSA *et al.*, 2009; SOUSA *et al.*, 2010).

Porém, para que todo o potencial da computação em nuvem possa ser explorado pelas organizações, é imprescindível assegurar a privacidade dos usuários e a segurança dos dados armazenados na nuvem. Não será possível atingir todo o potencial da computação em nuvem sem o desenvolvimento de estratégias que assegurem a proteção da privacidade dos dados de seus usuários (BRANCO *et al.*, 2016c).

Por outro lado, com a evolução dos dispositivos portáteis e o aumento da quantidade de serviços disponibilizados em nuvem, cresce também o volume de dados que precisa ser armazenado e processado. Para que o processamento desta grande quantidade de dados seja realizada em tempo viável, faz-se necessária a exploração de paradigmas de programação paralela e processamento distribuído. Contudo, desenvolver software para ambientes distribuídos é uma tarefa complexa, pois envolve uma série de conceitos e problemas que devem ser considerados pelos programadores, como concorrência, tolerância a falhas, distribuição de dados e balanceamento de carga. Infelizmente, dividir uma tarefa em subtarefas e então executá-las paralelamente em diversas unidades de processamento não é algo trivial. Adicionalmente, vale destacar que se o tamanho e a divisão das subtarefas não forem bem dimensionados, isso pode comprometer o desempenho da aplicação. Com a finalidade de facilitar este processo, surgiu o paradigma MapReduce, um modelo de programação paralela para processamento largamente distribuído de grandes volumes de dados, proposto inicialmente pela empresa *Google* em 2004 (DEAN; GHEMAWAT, 2004).

Este capítulo apresenta conceitos fundamentais sobre privacidade, segurança de dados, computação em nuvem e paradigma MapReduce. Estes conceitos se inter-relacionam quando analisamos o desafio de se preservar a confidencialidade dos dados armazenados na

nuvem. A necessidade de assegurar a privacidade das pessoas que produzem dados ou às quais os dados se referem é o que motiva os investimentos em técnicas de proteção de dados armazenados em nuvem, pois a descoberta de informações a partir do conhecimento de dados dispostos na nuvem, por parte de pessoas não autorizadas, representa o início da perda de privacidade (BRANCO *et al.*, 2017).

2.2 Privacidade

O estudo do tema privacidade abrange disciplinas da filosofia à ciência política, teoria política e legal, ciência da informação e, de forma crescente, engenharia e ciência da computação. Um consenso entre os pesquisadores é que privacidade é um assunto complexo. Inicialmente, privacidade é um conceito relacionado a pessoas. Trata-se de um direito humano, como liberdade, justiça ou igualdade perante a lei. Assim, a privacidade está relacionada ao interesse que as pessoas têm em manterem um espaço pessoal, sem interferências de outras pessoas ou organizações (BRANCO *et al.*, 2017).

O *American Institute of Certified Public Accountants* (AICPA) e o *Canadian Institute of Charted Accountants* (CICA) apresentam o conceito de privacidade da seguinte forma: “Privacidade é o conjunto de direitos e obrigações que indivíduos e organizações têm em relação à coleta, uso, retenção e divulgação de informações pessoais".

Existe necessidade de privacidade, independentemente do meio onde o sujeito esta inserido, a não ser que ele a desconheça ou não se importe com ela. A privacidade encontra uma barreira para a sua existência no mundo virtual, na facilidade da transmissão da informação. Quando conectado a uma rede de computadores, um dispositivo (computador, *notebook*, *tablet*, celular, etc) pode estar vulnerável a todo tipo de ataque externo. As informações contidas nesse dispositivo podem ser acessadas e divulgadas para o resto da rede.

A privacidade é um termo abrangente que envolve e utiliza a segurança da informação. Nesse caso, essa segurança é necessária para os dispositivos e para os meios de transmissão de informação. No mundo virtual, a segurança relaciona-se intensamente com os dados nele armazenados. Portanto, a segurança é utilizada para garantir nesse meio certos aspectos que são apresentados no ambiente físico, como confidencialidade, autenticação, integridade, não repudio, controle de acesso e disponibilidade (STALLINGS, 2010).

2.3 Segurança e Privacidade de Dados

O NIST define segurança computacional como sendo “a proteção conferida a um sistema de informação automatizado, a fim de atingir os objetivos propostos de preservação da integridade, disponibilidade e confidencialidade dos recursos do sistema de informação (incluindo *hardware*, *software*, *firmware*, informações/dados e telecomunicações)". Esta definição contém 3 conceitos chaves para segurança computacional: confidencialidade, disponibilidade e integridade (GUTTMAN; ROBACK, 1995).

1. Integridade é uma propriedade que assegura que a informação manipulada mantenha todas as características originais estabelecidas pelo seu proprietário, incluindo o controle de mudanças e a garantia do seu ciclo de vida (nascimento,manutenção e destruição). Assim. a integridade garante que a mensagem recebida é a mesma que foi transmitida e que não houve alteração de seu conteúdo de maneira intencional ou não. Integridade na computação em nuvem significa que os dados e sistemas serão acessados pelos usuários autorizados no tempo certo e de forma confiável. Algumas técnicas para assegurar integridade são serviços de filtragem de pacotes via *firewall*, gerenciamento de segurança por meio de configurações de protocolos, sistemas e tecnologias nas comunicações e serviços de detecção de intrusos.
2. Disponibilidade é uma propriedade que assegura que a informação esteja sempre disponível para o uso legítimo, ou seja, por aqueles usuários autorizados pelo proprietário da informação. Neste sentido, a disponibilidade se refere a elementos que criam estabilidade nas redes e sistemas, assegurando a conectividade quando necessária, permitindo que usuários autorizados possam acessar a rede e os sistemas. A confiabilidade de um sistema é definida como a sua probabilidade de estar operacional continuamente, durante um intervalo específico [0, t], dado que ele estava operacional em t = 0 (SIEWIOREK; SWARZ, 1992). A definição de confiabilidade de sistemas não leva em conta os tempos necessários para executar reparos ou processos de recuperação, necessários para restaurar o estado operacional de sistemas no evento de falhas. Apenas o tempo de operação contínua, sem interrupções, é considerado. A confiabilidade está diretamente relacionada ao tempo médio para falhar (*MTTF - Mean Time To Failure*). Quando sistemas ou seus elementos constituintes falham, normalmente, são reparados ou trocados por novos sistemas ou elementos, respectivamente. Assim sendo, o tempo de vida de um sistema ou elemento particular compreende períodos alternados de operação e falha. Considerando esse comportamento, pode-se medir os seguintes valores de recuperabilidade de sistemas: *MTBF - Mean Time*

Between Failures - tempo médio entre falhas e *MTTR - Mean Time To Repair* - tempo médio para reparo, onde *MTBF = MTTF + MTTR* (SIEWIOREK; SWARZ, 1992). Em resumo, o tempo médio entre falhas refere-se ao tempo transcorrido entre uma falha e outra (incluindo o tempo necessário para reparo/restauração) enquanto o tempo médio para reparo engloba o intervalo que inicia no instante em que uma falha é detectada até o instante em que o sistema ou seus elementos constituintes sejam completamente reparados ou trocados. A disponibilidade de um sistema é a sua probabilidade de estar operacional no instante de tempo t (SIEWIOREK; SWARZ, 1992). Utilizando-se *MTTF* e *MTTR*, pode-se definir a disponibilidade de sistemas como uma função da frequência de ocorrência de falhas e do tempo necessário para seu reparo ou troca, isto é, a disponibilidade é uma função direta do *MTTF* e *MTTR* (especificamente, disponibilidade é *MTTF/(MTTF+MTTR)*) ou disponibilidade é tempo operando / (tempo operando + tempo parado) (SIEWIOREK; SWARZ, 1992).

3. Confidencialidade é uma propriedade que limita o acesso a informação tão somente às entidades legítimas, ou seja, àquelas autorizadas pelo proprietário da informação. Neste sentido, confidencialidade diz respeito à prevenção contra a divulgação de dados não autorizados de forma intencional ou não intencional. Segundo Krutz (KRUTZ; VINES, 2010), confidencialidade em sistemas de nuvem envolve áreas como:
    - direitos de propriedade intelectual: são direitos sobre invenções, produções artísticas ou literárias que estão protegidos por leis de *copyright*;
    - canais secretos: são *links* de comunicação não intencionais ou não autorizados que permitem troca de informações. Podem ser criados por meio de mensagens de temporização ou uso não apropriado de mecanismos de armazenamento;
    - análise de tráfego de rede: é uma forma de violação de confidencialidade que pode ocorrer pela análise do volume, taxa, origem e destino de mensagens da rede;
    - encriptação: envolve criptografar mensagens para que só possam ser lidas por usuários autorizados que possuam a chave para descriptografar as mensagens;
    - inferência: normalmente está associada a segurança de banco de dados. Trata-se da capacidade de uma entidade utilizar e correlacionar informações em um nível de segurança para descobrir informações protegidas em um nível maior de segurança.

Por outro lado, segundo Stallings et. al. (STALLINGS, 2010), a privacidade dos dados assegura que os indivíduos controlam ou influenciam quais informações relacionadas a

eles podem ser coletadas e armazenadas por alguém e com quem elas podem ser compartilhadas. Neste caso, a privacidade de dados na Internet é a habilidade de um usuário ou organização controlar que informações eles revelam sobre si próprios na Internet, ou seja, controlar quem pode acessar qual informação e de que forma isto pode ocorrer.

Desta forma, confidencialidade e privacidade de dados são conceitos que mantêm uma forte relação. Ambos relacionam-se com a necessidade de assegurar que o indivíduo controle que informações relacionadas a ele podem ser divulgadas e quem pode acessá-las. Neste caso, pode-se entender a confidencialidade como uma ferramenta para se obter privacidade.

A coleta de dados na Internet sem o conhecimento ou a previa autorização do usuário afronta o direito à privacidade, podendo acarretar danos irreparáveis ao usuário, o qual dispõe de poucos meios para impedir tal prática. Derivado do direito à privacidade, podemos extrair o direito ao sigilo de informações. O não cumprimento deste direito decorrente da comercialização de Informações Pessoais Identificáveis *(Personally Identifiable Information-Personally Identifiable Information (PII))*, que são todas as informações relativas a uma determinada pessoa, desde características físicas até hábitos dos mais variados, de modo que do cruzamento desses dados seja possível traçar um verdadeiro perfil da respectiva pessoa, é um autêntico atentado à privacidade da pessoa humana. Assim sendo, as políticas de privacidade devem de forma bastante clara informar ao usuário sobre o tipo de informações que serão coletadas, o modo como será realizado a coleta dos dados, outrossim, como eles serão gerenciados, os motivos pelos quais os armazenam em seus bancos de dados, a possibilidade de cruzamento das informações coletadas junto a terceiros etc (VENKATARAMANAN; SHRIRAM, 2016).

Neste contexto, a proteção de dados está relacionada ao gerenciamento de informações pessoais. De modo geral, informações pessoais descrevem fatos, comunicações ou opiniões relacionadas a um indivíduo, as quais ele desejaria manter em segredo, controlando sua coleta, uso ou compartilhamento. Informações pessoais podem ser associadas a um indivíduo específico tais como nome, CPF, número do cartão de crédito, número da identidade. Algumas informações pessoais são consideradas mais sensíveis do que outras. Por exemplo, informações sobre saúde (registros médicos) são consideradas sensíveis em todas as jurisdições. Por exemplo, o conteúdo do prontuário médico só poderá ser revelado a terceiros se houver a autorização do paciente, conforme estabelece o artigo 5º da Resolução CFM n.º 1605/2000, ou se houver a anuência do Conselho Regional de Medicina da jurisdição, ex vi do artigo 8º do mesmo diploma, bem como autorização judicial. Informações biométricas e informações sobre avaliação de desempenho

no trabalho também são consideradas sensíveis. Este tipo de informação necessita de proteção adicional em relação à privacidade e segurança.

Na proporção em que grandes volumes de informações pessoais são transferidos para a nuvem, cresce a preocupação de pessoas e organizações sobre como estes dados serão armazenados e processados. O fato dos dados estarem armazenados em múltiplos locais, muitas vezes de forma transparente em relação à sua localização, provoca insegurança quanto ao grau de privacidade a que estão expostos.

2.4 Computação em Nuvem

Computação em nuvem *Cloud Computing* é uma tendência recente de tecnologia cujo objetivo é proporcionar serviços de Tecnologia da Informação (TI) sob demanda com pagamento baseado no uso. Tendências anteriores a computação em nuvem foram limitadas a uma determinada classe de usuários ou focadas em tornar disponível uma demanda específica de recursos de TI, principalmente de informática (BUYYA *et al.*, 2010). Apesar de ser uma tecnologia recente, o conceito de Computação em Nuvem é antigo. O termo *Cloud Computing* surgiu pela primeira vez em 1997, em uma palestra acadêmica ministrada por Ramnath Chellappa. Porém, o conceito é associado ao nome de John Mccarthy, pioneiro na tecnologia de Inteligência Artificial e criador da linguagem de programação LISP. Em 1960, ele disse que “a computação pode algum dia ser considerada como uma utilidade pública". A origem do termo vem dos diagramas das antigas redes de dados ISDN (*Integrated Services Digital Network*, ou rede de serviços digitais) e *Frame Relay*, projetadas pelas operadoras de telefonia. A interligação entre ambas era mostrada por desenhos de nuvens, para sinalizar algo que estava fora do alcance das empresas (HU, 2015).

Para que seja possível discutir em detalhes os principais aspectos relacionados à confidencialidade dos dados armazenados na nuvem, é preciso definir o que é computação em nuvem. Existem várias definições para computação em nuvem. Contudo, neste trabalho, a definição utilizada será a seguinte (HON *et al.*, 2011):

- a computação em nuvem fornece acesso flexível, independente de localização, para recursos de computação que são rapidamente alocados ou liberados em resposta à demanda;
- serviços (especialmente infraestrutura) são abstraídos e virtualizados, geralmente sendo alocados como um *pool* de recursos compartilhados com diversos clientes;
- tarifas, quando cobradas, geralmente são calculadas com base no acesso, de forma propor-

cional, aos recursos utilizados.

As aplicações da nuvem incluem processamento e integração de dados, serviços de armazenamento e comunicação, que são normalmente disponibilizados sob demanda e taxados com base no uso. A movimentação de dados e programas para a nuvem aumenta o poder de processamento e armazenamento dos usuários e oferece as seguintes vantagens:

1. Flexibilidade ilimitada para acessar milhões de bancos de dados e programas na nuvem, com possibilidade de combinar e customizar serviços.
2. Novas possibilidade de trabalho colaborativo entre os usuários da nuvem, por meio de compartilhamento *online* de dados e aplicações na nuvem.
3. Melhor confiabilidade e segurança no armazenamento dos dados na nuvem, em relação ao armazenamento dos dados nos dispositivos dos usuários, que podem ser perdidos ou roubados ou apresentarem falhas de hardware que resultem em perda de dados.
4. Ubiquidade: usuários podem acessar seus dados de qualquer lugar e a qualquer momento, desde que estejam conectados à internet.
5. Simplificação do *hardware* dos dispositivos dos usuários, que necessitam de menor poder de processamento e armazenamento de dados.

A computação em nuvem é composta de sete características fundamentais: três modelos de serviço e quatro abordagens de implantação (MELL; GRANCE, 2011). Os modelos de serviços de nuvem são disponibilizados normalmente nas seguintes plataformas:

1. Plataforma como Serviço – Platform as a Service (PaaS): provimento de recursos de *hardware*, tipicamente máquinas virtuais, que podem ser carregadas com o sistema operacional e *softwares* dos usuários.
2. Infraestrutura como Serviço – Infrastructure as a Service (IaaS): provimento de infraestrutura de *hardware*, incluindo plataformas virtuais, interconectividade, etc, onde as aplicações dos usuários podem ser instaladas.
3. *Software* como Serviço – Software as a Service (SaaS): disponibilização de toda infraestrutura de *hardware*, software para execução de aplicações de *software* na Internet pelos usuários de nuvem.

A implantação dos modelos pode seguir uma abordagem (i) pública – os recursos são dinamicamente disponibilizados na Internet para o público em geral, (ii) privada – os recursos são acessíveis apenas dentro de uma determinada corporação ou uma instituição cientifica, (iii) comunitária – recursos compartilhados por organizações com interesses comuns ou (iv) híbrida –

qualquer tipo de combinação entre as categorias anteriores.

O mecanismo de virtualização é amplamente utilizado na camada de infraestrutura da nuvem, porque permite a flexibilização do uso da camada de *hardware*. Máquinas virtuais proveem capacidade de processamento independente em máquinas isoladas, que podem ser instanciadas e destruídas sob demanda. Dessa forma, o ambiente de máquinas virtuais constitui uma base bastante adequada para a construção de infraestruturas de computação em nuvem (GROBAUER *et al.*, 2011). O modelo de serviço operacional, em conjunto com as tecnologias utilizadas para prover serviços do ambiente de computação em nuvem, apresenta diferentes níveis de riscos em comparação aos ambientes tradicionais de tecnologia de informação (BARDIN *et al.*, 2009). O provimento de recursos sob demanda para o processamento e armazenamento intensivo de dados está sujeito a falhas de segurança, abusos com relação à privacidade, violação de direitos autorais, etc.

As aplicações de maior sucesso e visibilidade na nuvem são serviços desenvolvidos para consumidores pessoais, tais como *e-mails*, redes sociais, e mundos virtuais (RAGHAVAN *et al.*, 2007). Para manter estes serviços, *terabytes* de dados são coletados, sendo que a maior parte são informações pessoais, que precisam de privacidade. As formas como os provedores de serviços de nuvem irão tratar os problemas de privacidade serão um fator decisivo para o desenvolvimento da tecnologia de computação em nuvem.

2.5 O Paradigma MapReduce

Com o constante aumento do volume de dados gerados a na rede mundial de computadores, diversos pesquisadores buscaram soluções para os problemas de processamento e armazenamento de grandes volumes de dados de maneira eficiente. Nesse contexto surgiu um novo modelo de programação denominado *MapReduce*, o qual foi inspirado nas primitivas *map* e *reduce* do Lisp e outras linguagens funcionais.

O modelo de programação *MapReduce* foi desenvolvido com o objetivo de possibilitar o processamento de grandes volumes de dados através da distribuição do processamento, inicialmente centralizado, em uma infraestrutura de *cluster*. Dessa forma, o processamento de dados é distribuído em vários computadores, onde cada um processa a porção de dados que lhe foi atribuída, omitindo do usuário toda a complexidade de paralelização, comunicação e gerenciamento da carga de trabalho.

Assim, a metodologia utilizada pelo modelo *MapReduce* para atingir o seu objetivo

consiste em aproveitar-se do paralelismo para dividir a carga de dados, ao invés de dividir as etapas de processamento. Em outras palavras, cada componente é responsável por processar completamente uma pequena porção de dados, ao invés de processar todos os dados em uma determinada etapa da computação.

A ideia principal por trás do *MapReduce* é mapear um conjunto de dados em uma coleção de tuplas (*chave*,*valor*), e, então, reduzir todas as tuplas com a mesma chave produzindo a saída final do processamento. Esta abordagem adota o princípio de abstrair toda a complexidade da paralelização de uma aplicação usando apenas as funções *Map* e *Reduce*. Essa ideia simples demonstrou ser um método eficaz de resolução de problemas usando programação paralela, uma vez que tanto *Map* quanto *Reduce* são funções sem estado associado e, portanto, facilmente paralelizável. Grande parte do sucesso do *MapReduce* foi alcançada pelo desenvolvimento de sistemas de arquivos distribuídos específicos para ele.

2.6 O Framework Spark

O *Apache Spark* é um *framework* de código aberto voltado para computação distribuída e utilizado comumente para cargas de trabalho de *Big Data*. O *Spark* foi desenvolvido no AMPLab da Universidade da Califórnia e posteriormente repassado para a *Apache Software Foundation*, que o mantém desde então. *Spark* provê uma interface para programação de *clusters* com paralelismo e tolerância a falhas. O *Apache Spark* utiliza o armazenamento em cache na memória e a execução otimizada para acelerar o seu desempenho, além de fornecer suporte para processamento em lotes, análise de *streaming*, aprendizado de máquina, bancos de dados gráficos e consultas *ad hoc* (KARAU; WARREN, 2017).

Por utilizar um mecanismo de execução baseado em um gráfico acíclico dirigido (DAG), o *Apache Spark* pode criar planos de consulta eficientes para transformações de dados. O *Apache Spark* também armazena dados de entrada, saída e intermediários na memória como conjuntos de dados resilientes e distribuídos (RDDs), o que permite um processamento rápido e sem custos de entrada e saída, aumentando o desempenho de cargas de trabalho iterativas e interativas. O *Apache Spark* fornece suporte de modo nativo a Java, Scala e Python. Além disso, é possível enviar consultas SQL ou HiveQL para o *Apache Spark* usando o módulo *Spark SQL*. Além de executar aplicativos, você pode usar a API do *Apache Spark* de modo interativo com Python ou Scala diretamente no *shell* do *Apache Spark* no seu *cluster*. O *Apache Spark* inclui várias bibliotecas para ajudar a criar aplicações de aprendizado de máquina (MLlib),

processamento de *streams* (*Spark Streaming*) e processamento de gráficos (*GraphX*). Essas bibliotecas estão solidamente integradas ao ecossistema do *Apache Spark*. Além disso, é possível usar estruturas de aprendizado profundo como *Apache MXNet*.

As funções *Spark* de alto nível podem ser utilizadas para realizar tarefas de processamento de dados em dados de diferentes tamanhos e estruturas. Por si só, o *Spark* não é uma solução de armazenamento de dados. Contudo, o *Spark* pode ser utilizado em conjunto com um sistema de armazenamento distribuído (por exemplo, HDFS, Cassandra ou S3) e um gerenciador de *cluster*. Neste caso, o sistema de armazenamento é usado para hospedar os dados processados com *Spark* e o gerenciador de *cluster* para orquestrar a distribuição de aplicativos *Spark* em todo o *cluster*. O *Spark* suporta atualmente três tipos de gerenciadores de *cluster*: Gerenciador de *cluster* independente, *Apache Mesos* e *Hadoop YARN*. O Gerenciador de *cluster* independente está incluso no *Spark*. Contudo, usar este gerenciador requer a instalação do *Spark* em cada nó do *cluster*.

2.6.1 RDD - Resilient Distributed Dataset

O *Spark* é construído em torno de uma abstração de dados chamada Sistemas de Dados Distribuídos Resilientes (RDDs). Os RDDs são uma representação de coleções distribuídas. É possível executar diversas operações sobre um RDD por meio das ações (*actions*) e transformações (*transformations*). As transformações são funções aplicadas em todo o conjunto de dados do RDD que produzem como saída um novo RDD. Como exemplos de transformações, pode-se citar: *map*,*join*, e *reduce*. As ações são operações que, a partir de um RDD, retornam apenas um resultado para o programa principal ou persistem algum dado no disco.

2.6.2 Componentes do Spark

O Spark tem diversos componentes para diferentes tipos de processamentos, todos construídos sobre o Spark Core, que é o componente que disponibiliza as funções básicas para o processamento, como, por exemplo, as funções *map*, *reduce*, *filter* e *collect*. A Figura 1 (KARAU; WARREN, 2017) ilustra esses componentes.

- Spark Streamming: possibilita o processamento de fluxos em tempo real;
- GraphX: realiza o processamento sobre grafos;
- SparkSQL: possibilitar utilizar a linguagem SQL na realização de consultas e processamento sobre os dados no Spark;

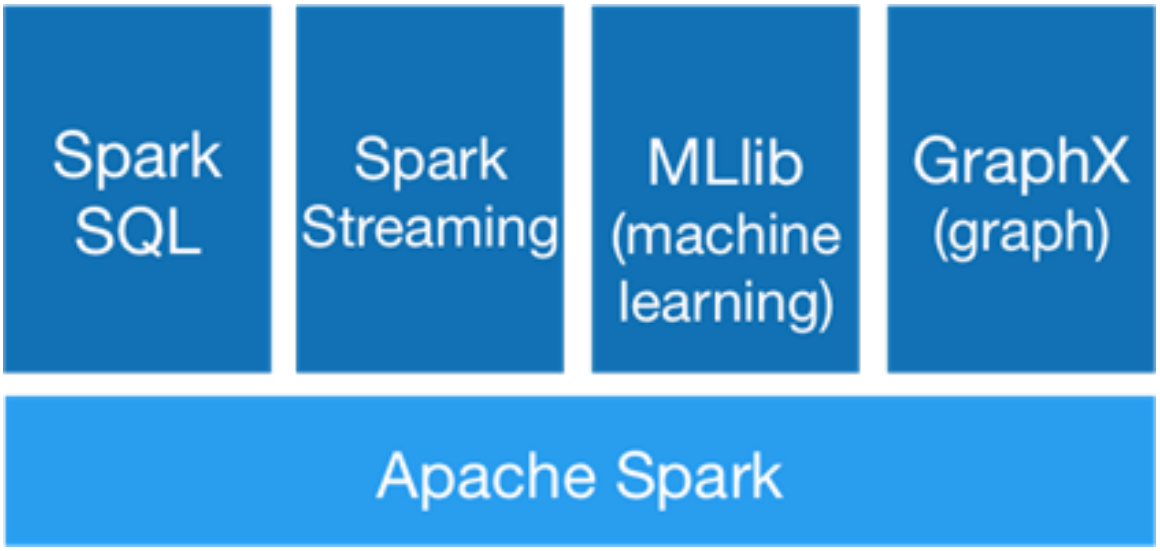

Figura 1 – Componentes de uma aplicação Spark

- MLlib: biblioteca de aprendizado de máquina, com deferentes algoritmos para as mais diversas atividades, como *clustering*.
- Spark Core: principal estrutura de processamento de dados no ecossistema *Spark*, tem APIs em Scala, Java, Python e R.

Vale destacar que uma das grandes vantagens do Spark é que todos os componentes funcionam integrados no próprio *framework*, diferentemente do *Hadoop*, onde é necessário utilizar ferramentas que se integram a ele, mas que são distribuídas separadamente, como o *Apache Hive*. Isso facilita bastante o desenvolvimento de aplicações de processamento de grandes volumes de dados. Adicionalmente, o Spark apresenta um desempenho muito superior ao Hadoop, chegando em alguns casos a apresentar uma performance quase 100x maior.

2.6.3 Arquitetura do Spark

A arquitetura do Spark é baseada no paradigma mestre e escravo, tendo o mestre responsável pela coordenação das execuções de aplicações no *cluster*. Ao submeter uma aplicação, o mestre executa o programa *driver* (*Driver Program*) o qual contém uma instância do *SparkContext* e os demais códigos referentes à aplicação (RDDs, transformações e ações). O *Spark* converte as aplicações em unidades denominadas *tasks* e organiza a execução dessas unidades nas máquinas escravas, denominadas executores (*executors* ou *workers*). Os executores são responsáveis por executar as tarefas que lhe foram concebidas e reportar os resultados para o programa *driver* no nó mestre. O Spark depende de um gerenciador de *cluster* (*Cluster Manager*) para inicializar os executores, o qual pode ser o *Hadoop YARN*, Mesos ou seu próprio gerenciador

de *cluster*. A Figura 2 ilustra a arquitetura do Spark.

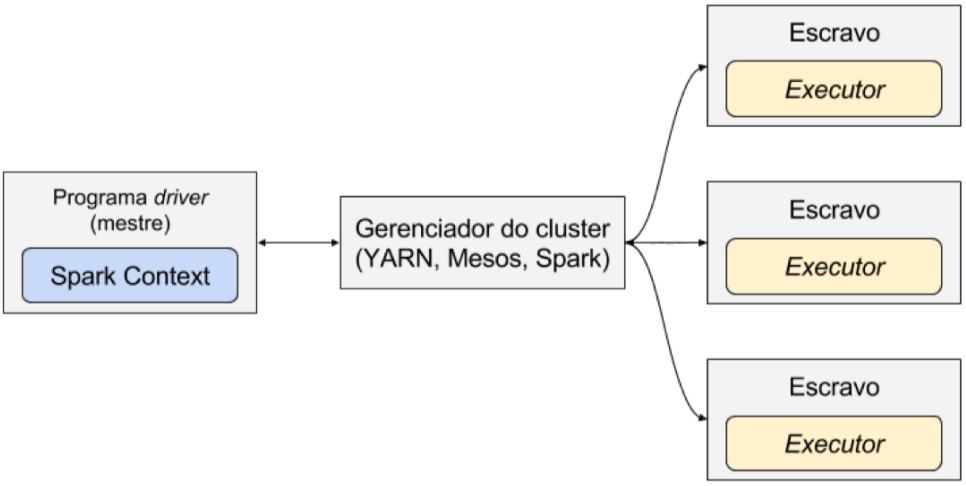

Figura 2 – Componentes de uma aplicação Spark

2.6.4 Modelo de Computação Paralela

O *Spark* permite que os usuários escrevam programas em um sistema de computação de *cluster*, os quais podem executar operações sobre dados em paralelo. O *Spark* representa grandes conjuntos de dados como RDDs - coleções de objetos distribuídos e imutáveis - que são armazenados nos executores (ou nós escravos). Os objetos que compõem RDDs são chamados de partições (*chunk*) e podem ser (mas não precisam ser) computados em diferentes nós de um sistema distribuído. O gerenciador de *cluster Spark* lida e distribui os executores *Spark* em um sistema distribuído de acordo com os parâmetros de configuração definidos pelo aplicativo *Spark*. O próprio mecanismo de execução distribui os dados em todos os executores para uma computação.

Em vez de avaliar cada transformação logo que especificada pelo programa, o *Spark* avalia os RDDs com *laziness*, executando as transformações no RDD somente quando os dados RDD finais precisam ser computados. O *Spark* pode manter um RDD carregado na memória nos nós do executor durante o ciclo de vida de um aplicativo *Spark* para acesso mais eficiente em cálculos repetitivos. Como eles são implementados no *Spark*, os RDDs são imutáveis, então transformar um RDD retorna um novo RDD em vez daquele previamente existente.

2.6.5 Avaliação Preguiçosa

Diversos sistemas para armazenamento em memória são baseados em atualizações "*fine-grained*"para objetos mutáveis, ou seja, chamadas para uma célula específica em uma tabela ao armazenar resultados intermediários. Em contraste, a avaliação de RDDs é completamente preguiçosa. *Spark* não começa a computar as partições até que uma ação seja chamada. Uma ação é uma operação *Spark* que retorna algo diferente de um RDD, desencadeando a avaliação de partições e possivelmente retornando algum resultado para um sistema não*Spark* (fora dos executores *Spark*), como por exemplo, trazendo os dados de volta para onde a aplicação está sendo executada, o *driver* (com operações como *count* ou *colect*), ou gravando dados em um sistema de armazenamento externo de armazenamento (como *copyToHadoop*). As ações acionam o agendador, que cria um gráfico acíclico direcionado (denominado DAG), com base nas dependências entre as transformações RDD. Em outras palavras, a Spark executa uma ação trabalhando para trás para definir a série de etapas a seguir para produzir cada objeto no conjunto de dados distribuído final (cada partição). Então, usando esta série de etapas, chamado plano de execução, o agendador calcula as partições faltantes para cada etapa até calcular o resultado.

A avaliação preguiçosa permite que o *Spark* combine operações que não requerem comunicação com o *driver* (chamadas transformações com dependências um para um) para evitar múltiplas passagens através dos dados. Por exemplo, suponha que um programa *Spark* chame um *map* e uma função *filter* no mesmo RDD. O *Spark* pode enviar as instruções para o *map* e o *filter* para cada executor. Então, o *Spark* pode executar o *map* e o *filter* em cada partição, o que requer acessar os registros apenas uma vez, em vez de enviar dois conjuntos de instruções e acessar cada partição duas vezes. Isto reduz teoricamente a complexidade computacional pela metade.

O paradigma de avaliação preguiçoso do *Spark* não é apenas eficiente, mas também é fácil de implementar. A estratégia inteligente de avaliação preguiçosa do *Spark* nos permite expressar a mesma lógica utilizando uma quantidade menor de linhas de código.

2.6.6 Tolerância a Falhas

O *Spark* é tolerante a falhas, o que significa que ele não irá perder os dados ou retornar resultados imprecisos no caso de uma falha em uma máquina (*host*) ou falha na rede de comunicação. O método único de tolerância a falhas do *Spark* é alcançado porque cada

partição dos dados contém as informações de dependência necessárias para recalcular a partição. A maioria dos paradigmas de computação distribuídos que permitem aos usuários trabalhar com objetos mutáveis fornecem tolerância a falhas ao registrar atualizações ou duplicar dados em máquinas. Em contraste, o *Spark* não precisa manter um registro de atualizações para cada RDD ou registrar as etapas intermediárias reais, já que o RDD contém todas as informações de dependência necessárias para replicar cada uma de suas partições. Assim, se uma partição for perdida, o RDD possui informações suficientes sobre sua linhagem para recalculá-la, e essa computação pode ser paralelizada para tornar a recuperação mais eficiente.

2.6.7 Memória: Persistência e Gerenciamento

A vantagem de desempenho do *Spark* sobre outros *frameworks* MapReduce é maior em cenários que envolvem cálculos repetidos. Grande parte desse aumento de desempenho deve-se ao uso da persistência em memória. Ao invés de escrever para o disco entre cada passagem através dos dados, o *Spark* tem a opção de manter os dados nos executores carregados na memória. Dessa forma, os dados em cada partição estão disponíveis na memória sempre que precisa ser acessado. O *Spark* oferece três opções para o gerenciamento de memória: na memória com dados não serializados, na memória com dados serializados, e no disco. Cada um tem diferentes vantagens em relação a espaço e tempo.

Na Memória com Dados Não Serializados:

A maneira mais intuitiva de armazenar dados em RDDs é utilizar os objetos Java (não serializados) que são definidos pelo programa de *driver*. Essa forma de armazenamento na memória é a mais rápida, uma vez que evita o tempo gasto na serialização. Contudo, pode não ser a mais eficiente em relação ao espaço (tamanho) de memória utilizado, uma vez que exige que os dados sejam armazenados como objetos.

Dados Serializados:

Usando a biblioteca de serialização Java padrão, os objetos *Spark* são convertidos em fluxos de *bytes* à medida que são movidos pela rede. Essa abordagem pode ser mais lenta. Porém, muitas vezes é mais eficiente em relação à utilização da memória, pois permite ao usuário escolher uma representação mais eficiente. Ainda que a serialização em Java seja mais eficiente, em termos de espaço, que os objetos completos, a serialização *Kryo* pode ser ainda mais eficiente.

Armazenamento em Disco:

RDDs, cujas partições são muito grandes para serem armazenadas na memória

principal dos executores, podem ser gravadas em disco. Esta estratégia é, obviamente, mais lenta para cálculos repetidos, mas pode ser mais tolerante a falhas para longas sequências de transformações e pode ser a única opção viável para computações envolvendo grandes volumes de dados.

2.6.8 Funções em RDDs: Transformações versus Ações

Existem dois tipos de funções definidas em RDDs: ações e transformações. As ações são funções que retornam algo que não é um RDD, incluindo um efeito colateral. Já as transformações são funções que retornam outro RDD. Cada programa *Spark* deve conter uma ação, uma vez que as ações trazem as informações de volta para o *driver* ou gravam os dados em um armazenamento estável. As ações são o que forçam a execução de um programa *Spark*. As chamadas persistentes também forçam a avaliação, mas geralmente não marcam o final do *Job Spark*. As ações que trazem dados de volta para o *driver* incluem coletar, contar, etc.

2.6.9 Shuffle

Shuffle significa a movimentação de dados gerada entre os nós do *cluster* para calcular o resultado de uma determinada operação. Imagine que se tenha uma lista de registros de detalhes da chamada telefônica em uma tabela e que se deseje calcular a quantidade de chamadas que aconteceu a cada dia. Desta forma, haveria a configuração do “dia"como sua chave, e para cada registro (ou seja, para cada chamada), haveria a emissão de “1"como um valor. Depois disso, os valores para cada chave seriam resumidos, o que seria uma resposta para a pergunta - quantidade total de registros para cada dia. Mas quando se armazenam os dados através do *cluster*, como é possível resumir os valores da mesma chave armazenada em diferentes máquinas? A única maneira de realizar esta tarefa consiste em fazer com que todos os valores da mesma chave estejam na mesma máquina. Depois disso será possível resumi-los. Logicamente, esta movimentação de dados deve ser evitada para se conseguir um melhor desempenho.

2.7 Scala

Scala é uma linguagem de programação multi-paradigma moderna, projetada para expressar padrões de programação comuns de forma concisa, elegante e segura, descrita em (WAMPLER; PAYNE, 2014). Ela integra perfeitamente funcionalidades de linguagens orientadas

a objetos e funcionais. Scala é uma linguagem pura orientada a objetos, no sentido de que cada valor é um objeto. Tipos e comportamentos de objetos são descritos por classes e traços. As classes são estendidas por subclasse e um mecanismo de composição flexível baseado em combinação como uma substituição limpa para múltiplas heranças.

Scala também é uma linguagem funcional no sentido de que cada função é um valor. Scala fornece uma sintaxe leve para definir funções anônimas, dando suporta funções de ordem superior, permitindo que as funções sejam aninhadas. Além disso, a noção de padronização da linguagem Scala se estende naturalmente ao processamento de dados XML. Esses recursos tornam Scala ideal para desenvolver aplicativos como serviços da Web.

Ao usar uma linguagem como Scala, é mais fácil escalar seu pequeno problema para um grande problema com Spark, cuja API é quase de 1 pra 1, em que uma coleção em Scala pode corresponder exatamente à uma RDD em *Spark*. Ou seja, trabalhando em Scala, em um estilo funcional, você pode escalar rapidamente seu problema de um nó para dezenas, centenas ou mesmo milhares utilizando o *Spark*.

2.8 Conclusão

Este capítulo apresentou as principais definições relacionadas a privacidade, destacando, principalmente, a relação entre privacidade e confidencialidade de dados. Além disso, discutimos os conceitos de computação em nuvem, o paradigma MapReduce, o *framework Apache Spark* e a linguagem de programação Scala.

3 TRABALHOS RELACIONADOS

3.1 Introdução

Nos últimos anos, várias propostas para assegurar a confidencialidade dos dados armazenados em ambiente de computação em nuvem tem sido apresentadas, tais como as encontradas em (RYAN, 2011), (CHEN; ZHAO, 2012), (NIMGAONKAR *et al.*, 2012), (STEFANOV E. SHI, 2013), (LI *et al.*, 2013), (YANG *et al.*, 2013), (JUNG *et al.*, 2013), (YEH, 2013). Mais recentemente, a abordagem QSM-Extraction, principal referência neste trabalho, foi proposta nos trabalhos de (BRANCO *et al.*, 2014), (BRANCO *et al.*, 2016a), (BRANCO *et al.*, 2016b).

As próximas seções apresentam as recentes pesquisas sobre o uso isolado e combinado das técnicas de criptografia, fragmentação e decomposição para aumento da confidencialidade dos dados armazenados na nuvem.

3.2 Criptografia

Uma quantidade significativa de pesquisas basearam-se em técnicas criptográficas para assegurar a confidencialidade de dados armazenados e processados em ambiente de computação em nuvem (POPA *et al.*, 2011; NING *et al.*, 2014). A maior parte destas pesquisas têm assumido que os dados devem estar inteiramente criptografados, concentrando-se no aprimoramento das técnicas de execução de consultas (CIRIANI *et al.*, 2010). Em (CESELLI *et al.*, 2005), os autores discutem diferentes estratégias para avaliação da inferência da exposição de dados criptografados acrescidos de informações de indexação, mostrando que mesmo um número limitado de índices pode favorecer muito a tarefa de um atacante que pretende violar a confidencialidade dos dados.

A criptografia é uma ferramenta útil para aumentar a confidencialidade de dados sensíveis. Entretanto, quando os dados são encriptados, a realização de consultas se torna um desafio. Assim, embora a encriptação de dados proporcione confidencialidade, as cifras produzidas são muito menos convenientes para uso do que os dados originais. Quando utilizada com bancos de dados relacionais, a criptografia cria dois grandes problemas. O primeiro problema é que os bancos relacionais requerem que os tipos de dados sejam definidos antes do seu armazenamento. O segundo problema é que consultas ou funções não podem ser executadas sobre dados criptografados. Não é possível avaliar faixas de datas ou fazer comparações de

valores em dados criptografados. As estruturas de índice também não podem ser utilizadas. Adicionalmente, os métodos baseados em criptografia precisam incluir estratégias de geração e distribuição de chaves (TIAN; ZHANG, 2012). Porém, existem várias desvantagens relacionadas com a gestão de chaves criptográficas, tais como:

1. A necessidade de guardar as chaves pelo igual período de tempo em que os dados permanecerem criptografados.
2. A atribuição ou a revogação de chaves para o acesso aos dados por parte dos usuários.
3. A necessidade de manter múltiplas cópias encriptadas do mesmo arquivo, para acesso multi-usuário utilizando chave-pública.

3.3 Fragmentação

A primeira abordagem que propõe o armazenamento de dados não criptografados, assegurando ao mesmo tempo uma série de restrições de confidencialidade foi apresentada em (AGGARWAL, 2005). Neste trabalho, os autores supõem que os dados sejam divididos em dois fragmentos, os quais serão armazenados em dois provedores de serviço de nuvem honestos-mas-curiosos, que nunca compartilharão informações entre si e recorre à encriptação quando estes dois fragmentos não são suficientes para impor restrições de confidencialidade.

Em (CIRIANI *et al.*, 2009; CIRIANI *et al.*, 2010), os autores resolvem o problema da confidencialidade, propondo uma solução que primeiro divide os dados a serem protegidos em vários (possivelmente mais de dois) fragmentos diferentes de modo a quebrar as associações sensíveis entre atributos e minimizar a quantidade de atributos representados apenas em formato criptografado. Esses fragmentos podem ser armazenados em diferentes servidores. A heurística proposta para descobrir esses fragmentos apresenta um tempo de custo de computação em tempo polinomial, ao mesmo tempo que é capaz de recuperar soluções próximas ao ideal. Uma solução próxima ao ideal é aquela em que a relação entre o custo $\mathbf{C}$ da heurística proposta e o custo $\mathbf{C}^*$ de uma solução que retorna todos os resultados possíveis (solução ótima) é menor do que um determinado parâmetro $\mathbf{p}$. Caso contrário, não é possível realizar qualquer avaliação do resultado da heurística.

Em (XU *et al.*, 2015), os autores propõem um método eficiente baseado em pesquisa em grafo para o problema de fragmentação com restrições de confidencialidade, que obtém resultados próximos ao ideal. (SAYI *et al.*, 2012) estendeu o trabalho de Ciriani, propondo o uso de algoritmos de coloração de grafos para determinar quais dados deveriam ser armazenados na

nuvem e quais deveriam ficar sobre a guarda do proprietário dos dados.

3.4 Criptografia e Fragmentação

O trabalho apresentado em (CIRIANI *et al.*, 2009) propõe um novo paradigma para preservação da confidencialidade dos dados sob a guarda de terceiros, que dispensa uso de criptografia, liberando assim o proprietário do dado do ônus de gestão de chaves. A ideia básica por trás deste mecanismo é envolver o proprietário do dados no armazenamento de atributos sensíveis. Assim sendo, para cada associação sensível, o proprietário deve armazenar localmente pelo menos um atributo. Os atributos restantes são armazenados, em formato não criptografado, do lado do servidor de nuvem. Com este processo de fragmentação, uma relação original R é então quebrada em dois fragmentos, chamados F_o e F_s, que são armazenados na infraestrutura de TI do proprietário do dado e no servidor de nuvem, respectivamente. (WIESE, 2010) faz uma extensão da abordagem de “apenas fragmentação vertical"e propõe utilizar fragmentação horizontal para filtrar linhas confidenciais para serem armazenadas de forma confidencial na infraestrutura do proprietário do dado. Em (REKATSINAS *et al.*, 2013), os autores apresentam SPARSI, um *framework* teórico para particionamento de dados sensíveis em múltiplos servidores adversários não coniventes. Eles introduzem o problema de particionamento de dados entre “respeitadores da privacidade", onde um conjunto de dados sensíveis deve ser dividido entre k partes não confiáveis (adversários). O objetivo é o de maximizar a utilidade derivada do particionamento e distribuição do conjunto de dados, enquanto minimiza a quantidade total de informação sensível divulgada. Solução de particionamento entre “respeitadores da privacidade"é, em geral, NP-difícil.

Em (SAMARATI; VIMERCATI, 2010) os autores discutem as principais questões a serem abordadas nos serviços de armazenamento em nuvem, que vão desde a confidencialidade dos dados até a utilidade dos dados. Eles mostram as principais direções de pesquisa que estão sendo investigadas para fornecer efetiva confidencialidade de armazenamento e consulta de dados. A pesquisa apresentada em (JOSEPH *et al.*, 2013) endereça algumas abordagens para garantir a confidencialidade dos dados em serviços de armazenamento em nuvem não confiáveis. Em (SAMARATI, 2014), os autores discutem os problemas da garantia adequada confidencialidade de dados e privacidade dos usuários na nuvem, e ilustram as possíveis soluções para eles.

3.5 Decomposição

Em (BRANCO *et al.*, 2014; BRANCO *et al.*, 2016a; BRANCO *et al.*, 2016b), os autores apresentam uma nova estratégia, denominada QSM-EXTRACTION, para assegurar a confidencialidade de dados em serviços de armazenamento em nuvem. Esta estratégia é inédita e única quanto a metodologias de decomposição. A ciência por trás dessa abordagem utiliza conceitos da Doutrina do Ser de Hegel. A estratégia QSM-EXTRACTION baseia-se na fragmentação de um arquivo digital em fragmentos denominados objetos de informação, na decomposição desses objetos por meio da extração de suas características (Qualidade, Quantidade e Medida) e na dispersão dessas características em diferentes serviços de armazenamento em nuvem, permitindo a posterior recuperação desses dados sem perda de informação. A finalidade da estratégia proposta é inviabilizar a reconstrução do arquivo original por parte de um provedor de nuvem que possui apenas parte das características dos objetos de informações que compõem este arquivo. Desta forma, assegura-se a confidencialidade dos dados armazenados em nuvem e, por conseguinte, a privacidade dos proprietários desses dados.

A estratégia QSM-EXTRACTION aplica-se a arquivos digitais que possuem as seguintes características:

- os arquivos armazenam as informações por meio de símbolos que são representados na forma de *bytes*, ou seja, os símbolos contidos no arquivo são representados por arranjos de 8 *bits* que podem formar 256 números na base 2, de 0_{10} $(00000000)_2$ a 255_{10} $(11111111)_2$;
- o tamanho de cada arquivo deve ser maior ou igual a 256 *bytes*, sendo que não há restrição de limite de tamanho máximo;
- os arquivos estão compactados. A compactação, ou compressão sem perdas, é amplamente utilizada, com o objetivo de reduzir os custos de armazenamento. Alguns exemplos de arquivos compactados são documentos (.doc, .pdf, .ps), arquivos de imagem (.jpg, .bmp, .eps), arquivos de som (.wav, .mp3, .wma), arquivos de vídeo (.wmv, .avi., .flv), dentre outros formatos (.zip, .rar. .lhz).

A ideia principal da estratégia QSM-EXTRACTION foi inspirada nas ideias do filósofo alemão Georg Wilhelm Friedrich Hegel, publicadas no livro **Enciclopédia das Ciências Filosóficas** em 1817. A Doutrina do Ser trata da Lógica do Ser aborda três conceitos principais: determinidade (qualidade), a grandeza (quantidade) e a medida. O estudo da lógica de Hegel sobre a formação dos objetos foi fonte de inspiração para o desenvolvimento da técnica de extração das características de qualidade, quantidade e medida de um arquivo digital, visto como

um objeto de informação.

As informações contidas nos arquivos digitais são representadas sob a forma de *byte*s. Existem várias codificações propostas para os arquivos digitais, uma das mais conhecidas e utilizadas é o Código Padrão Americano para o Intercâmbio de Informação (*American Standard Code for Information Interchange - ASCII*), apresentado no Apêndice E, que codifica um conjunto de 128 sinais: 95 sinais gráficos (letras do alfabeto latino, sinais de pontuação e sinais matemáticos) e 33 sinais de controle, utilizando portanto apenas 7 bits para representar todos os seus símbolos. Outro padrão internacional para codificação de informações em arquivos digitais é o padrão **Unicode**, que foi criado em 1989 para permitir aos computadores representar e manipular, de forma consistente, texto de qualquer sistema de escrita existente. O **Unicode** define dois métodos de mapeamento de códigos **Unicode** em códigos de implementação (CONSORTIUM, 2000):

- UTF-8 (*Unicode Transformation Format*), que utiliza entre um e quatro bytes por código.
- UTF-16 que representa o conjunto UCS (*Universal Character Set*), que utiliza uma codificação 16-bit de largura variável. Ela pode incluir uma ou duas palavras 16-bit para suportar outros caracteres.

A QSM-EXTRACTION trata as propriedades de qualidade, quantidade e medida dos arquivos digitais, considerando a representação de 1 símbolo por *byte*. Neste caso, o arquivo digital é tratado como um conjunto de **n** partes, onde cada parte possui 256 *bytes* de tamanho. Caso a parte final do arquivo seja menor do que 256 *bytes*, esta será complementada com *bytes* quaisquer, para completar a sequência de 256 bytes. Cada parte de 256 *bytes* sequenciais de um determinado arquivo será denominada de objeto de informação, que chamaremos de **iobjeto**. Um objeto de informação pode armazenar entre 1 e 256 símbolos diferentes. De um modo geral, as três características de um objeto de informação são as seguintes:

- **Qualidade** representa os diferentes símbolos (*bytes*) que compõem o arquivo digital;
- **Quantidade** representa a quantidade de vezes em que cada *byte* está presente no objeto de informação;
- **Medida** representa a informação sobre a ordem em que os *bytes* estão dispostos sequencialmente no objeto de informação.

A seguir, apresentamos as definições formais dos conceitos de objeto de informação e suas características de Qualidade, Quantidade e Medida:

Definição 1: (objeto de informação). Um objeto de informação é uma fragmento de um arquivo digital contendo 256 *bytes* sequenciais. Mais formalmente, um objeto de informação

é definido da seguinte forma: Seja um arquivo F = $< b_1, b_2, \cdots, b_n >$, em que b_i é um *byte*, 1 $\leqslant i \leqslant$ n e n $\geqslant$ 256. Um objeto de informação iOBJ = $< b_j, b_{j+1}, \cdots, b_{j+255} >$ onde j $\geqslant$ 1, j+255 $\leqslant$ n e iOBJ $\subset$ F.

Para exemplificar a definição de objeto de informação, considere um arquivo **F** contendo 768 *bytes*. **F** = $< b_1, b_2, \cdots, b_{768} >$, em que b_i é um *byte*. Neste caso, o arquivo **F** será fragmentado em 3 objetos de informação ($iOBJ_1$, $iOBJ_2$ e $iOBJ_3$), onde $iOBJ_1$ possui os primeiros 256 *bytes* sequenciais (de b_1 até b_{256}), $iOBJ_2$ possui os próximos 256 *bytes* sequenciais (de b_{257} até b_{512}) e $iOBJ_3$ possui os últimos 256 *bytes* sequenciais (de b_{513} até b_{612}).

Definição 2: (Qualidade). Qualidade é o conjunto de *bytes* distintos que compõem um determinado objeto de informação. Seja iOBJ um objeto de informação, **Q(iOBJ)** denota a propriedade da qualidade do iOBJ. **Q(iOBJ)** é um vetor ordenado que contem m *bytes* diversos presentes em iOBJ. Mais formalmente, **Q(iOBJ)** = $\{b_1, b_2, b_3, \cdots, b_m\}$ tal que $1 \leqslant b_i \leqslant 256$ e $i \neq j \rightarrow b_i \neq b_j$, onde b_i é um *byte* existente em iOBJ.

Definição 3: (Quantidade). Quantidade contém informações sobre o número de vezes que cada *byte* diverso aparece em um determinado objeto de informação. Seja iOBJ um objeto de informação, **S(iOBJ)** denota a propriedade da quantidade (extensão) do iOBJ. **S(iOBJ)** é um vetor de 256 posições que contém, para cada *byte* diverso b_j existente em **Q(iOBJ)**, o número de vezes que b_i aparece em iOBJ. Mais formalmente, **S(iOBJ)** = $\{s_1, s_2, s_3, \cdots, s_m\}$ tal que $1 \leqslant s_i \leqslant 256$, onde s_i representa o número de vezes que b_i aparece em iOBJ.

Definição 4: (Medida). Seja iOBJ um objeto de informação, M(iOBJ) denota a propriedade da medida de iOBJ. **M(iOBJ[256][256])** contém, para cada *byte* diverso b_j presente em **Q(iOBJ)**, um vetor m_{b_j} que armazena as posições nas quais o *byte* b_j aparece em iOBJ. Mais formalmente, M(iOBJ) = $\{m_{b_1}, m_{b_2}, \cdots, m_{b_m}\}$, tal que, m = 256 e $1 \leqslant size(m_{b_i}) \leqslant 256$.

A Medida armazena as posições ocupadas pelos *bytes* no iOBJ. Essas informações estão em uma matriz esparsa de 256 linhas por 256 colunas. As linhas representam a qualidade dos *bytes* existentes no objeto de informação e as colunas representam a quantidade de posições ocupadas pelos *bytes* no objeto de informação. Caso exista apenas um *byte* que se repete 256 vezes dentro do objeto de informação, os 256 elementos da linha da matriz da Medida correspondente a este *byte* ficarão totalmente preenchidos, e as demais linhas da matriz ficarão vazias. Na situação oposta, caso todos os 256 *bytes* do objeto de informação sejam diferentes uns dos outros, a primeira coluna da matriz da Medida ficará completamente preenchida, ficando vazias, as demais colunas da matriz.

Um visão geral da estratégia QSM-EXTRACTION é apresentada na Figura 3. Neste cenário temos 5 atores: um cliente, que deseja armazenar seus arquivos pessoais na nuvem pública, três provedores de serviços de armazenamento de dados em nuvens públicas e uma Terceira Parte Confiável (TPC) que é responsável pelo processamento dos algoritmos que compõem a estratégia QSM-EXTRACTION, os quais irão assegurar a confidencialidade dos dados, além do controle das comunicações entre o cliente e os provedores de armazenamento em nuvem.

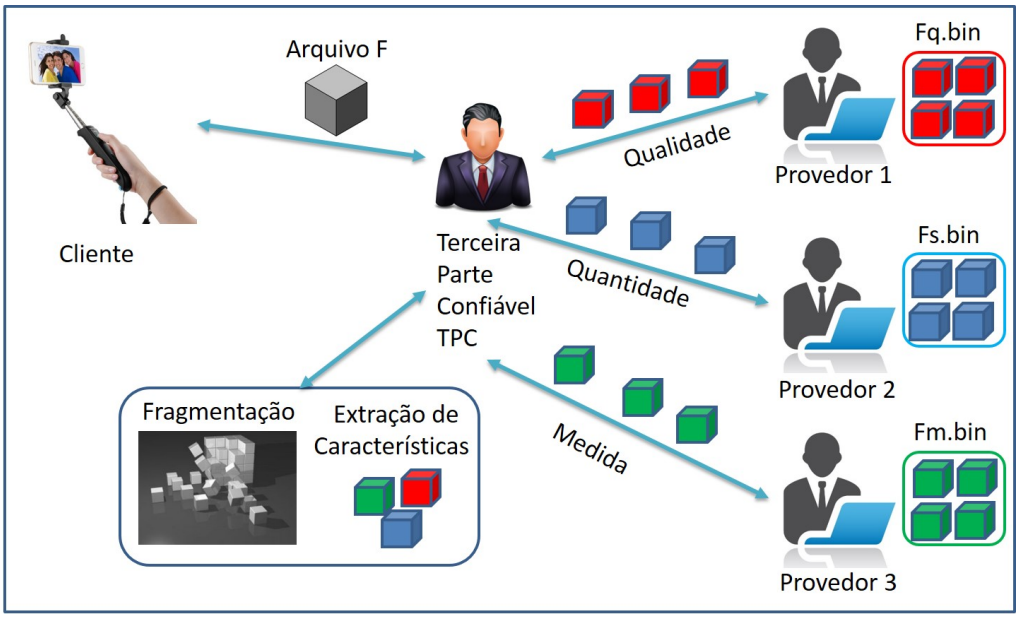

Figura 3 – Visão Geral do Armazenamento de Dados nas Nuvens

As fases e etapas do processo de criação e armazenamento dos objetos de informação na infraestrutura de armazenamento da nuvem são as seguintes:

1. Fase de Fragmentação
    - Fragmentação do arquivo em objetos de informação;
2. Fase de Decomposição
    - Extração dos atributos de Qualidade, Quantidade e Medida dos objetos de informação;
    - Anonimização da medida;
    - Anonimização da qualidade;
    - Anonimização da quantidade
    - Compactação da medida;
3. Fase de Dispersão
    - Envio dos três arquivos contendo as características dos objetos de informação que

compõem o arquivo **F** para três servidores distintos na nuvem.

- Armazenamento das informações sobre a qualidade dos objetos de informação no arquivo **Fq.bin**, por exemplo na nuvem 1;
- Armazenamento das informações sobre a quantidade dos objetos de informação no arquivo **Fs.bin**, por exemplo na nuvem 2;
- Armazenamento das informações sobre a medida dos objetos de informação no arquivo **Fm.bin**, por exemplo na nuvem 3.

A seguir, ilustramos a execução da estratégia QSM-EXTRACTION. Seja um arquivo texto contendo o seguinte conteúdo:

O rato roeu a roupa do rei de roma.

Inicialmente, calculamos Qualidade, Quantidade e Medida. A Qualidade se refere ao número decimal em código American Standard Code for Information Interchange (ASCII) de cada caractere do texto. O primeiro caractere "O"corresponde ao decimal 79 na tabela ASCII. Ele ocorreu apenas uma vez no texto, na posição inicial "0". Seguindo esse raciocínio, estabelecemos o conteúdo do parâmetro Qualidade como:

Qualidade(Q) = {79(O), 32(espaco), 114(r), 97(a), 116(t), 111(o), 101(e), 117(u), 112(p), 100(d), 105(i), 109(m), 46(.)}.

Os parâmetros Quantidade e Medida contêm, respectivamente, informações referentes às quantidades de ocorrência de cada caractere e às posições de suas correspondentes ocorrências. Para o exemplo acima, seus conteúdos são dados como:

Quantidade(S) = {1, 8, 5, 4, 1, 4, 2, 2, 1, 2, 1, 1, 1},

Medida(M) = {{0}, {1,6,11,13,19,22,26,29}, {2,7,14,23,30}, {3,12,18,33}, {4}, {5,8,15,21,31}, {9,24}, {10,16}, {18}, {20,27}, {25}, {32}, {34}}.

A Tabela 1 sintetiza e relaciona os resultados das propriedades Qualidade (Q), Quantidade (S) e Medida (M), obtidos para o exemplo anterior.

3.6 Conclusão

Este capítulo descreveu os trabalhos relacionados à proteção da confidencialidade dos dados armazenados na nuvem que utilizam técnicas de criptografia, fragmentação e decomposição, isoladamente ou em conjunto.

Tabela 1 – Exemplo de execução QSM-Extraction

Qualidade(Q)	Quantidade(S)	Medida(M)
79	1	0
32	8	1,6,11,13,19,22,26,29
114	5	2,7,14,23,30
97	4	3,12,18,33
116	1	4
111	4	5,8,15,21,31
101	2	9,24
117	2	10,16
112	1	18
100	2	20,27
105	1	25
109	1	32
46	1	34

4 A ESTRATÉGIA PDQSM-EXTRACTION

Este capítulo descreve uma versão paralela e distribuída da estratégia QSM-EXTRACTION, denominada pdQSM-EXTRACTION. A QSM-EXTRACTION baseia-se na fragmentação de um arquivo digital em fragmentos denominados objetos de informação, na decomposição desses objetos por meio da extração de suas características e na dispersão dessas características em diferentes serviços de armazenamento em nuvem, permitindo a posterior recuperação desses dados sem perda de informação.

A estratégia QSM-EXTRACTION busca inviabilizar a reconstrução do arquivo original por parte de um provedor de nuvem que possui apenas parte das características dos objetos de informações que compõem este arquivo. Assim, assegura-se a confidencialidade dos dados armazenados em nuvem e, por conseguinte, a privacidade dos proprietários desses dados.

Todavia, apesar da estratégia QSM-EXTRACTION ter sido concebida para o ambiente de computação em nuvem, esta adota uma abordagem de execução centralizada, o que pode comprometer o desempenho da etapa de decomposição. Com a finalidade de superar esta limitação, a pdQSM-EXTRACTION explora a utilização do Paradigma MapReduce na etapa de extração das características dos objetos de informação. A pdQSM-EXTRACTION foi implementada em linguagem Scala utilizando-se o *framework Apache Spark*. Nossa hipótese é que, a partir de um determinado tamanho de arquivo, a estratégia pdQSM-EXTRACTION se torna mais eficiente que a QSM-EXTRACTION.

A Figura 4 fornece uma visão geral da estratégia pdQSM-EXTRACTION, a qual explora o Paradigma MapReduce na etapa de decomposição, ou seja, com o objetivo de paralelizar a extração das características dos objetos de informação. Assim, a computação das características (qualidade, medida e quantidade) dos objetos de informação que compõem um arquivo digital é realizada de forma paralela.

Na estratégia pdQSM-EXTRACTION, inicialmente, um arquivo digital *AO* é fragmentado em partições denominadas *chunks*. Um *chunk* pode representar um ou mais objetos de informação. Este processo de fragmentação é conduzido pelo *framework Apache Spark*. Em seguida, essas partições (*chunks*) são distribuídas entre diferentes nós de um *cluster*. Essa etapa também é orquestrada pelo *Spark*. Posteriormente, cada nó (*Mapper*) do *cluster*, em paralelo, irá decompor as partições sob sua responsabilidade, por meio de um processo *Map*, computando as características de qualidade, quantidade e medida para cada partição (*chunk*). Por fim, essas características são enviadas para um nó do *cluster* (chamado *Reducer*) que irá combinar tais

informações, por meio de um processo *Reduce*, com a finalidade de produzir como saída três arquivos, contendo, respectivamente, as características de qualidade, quantidade e medida, para o arquivo digital *AO* como um todo.

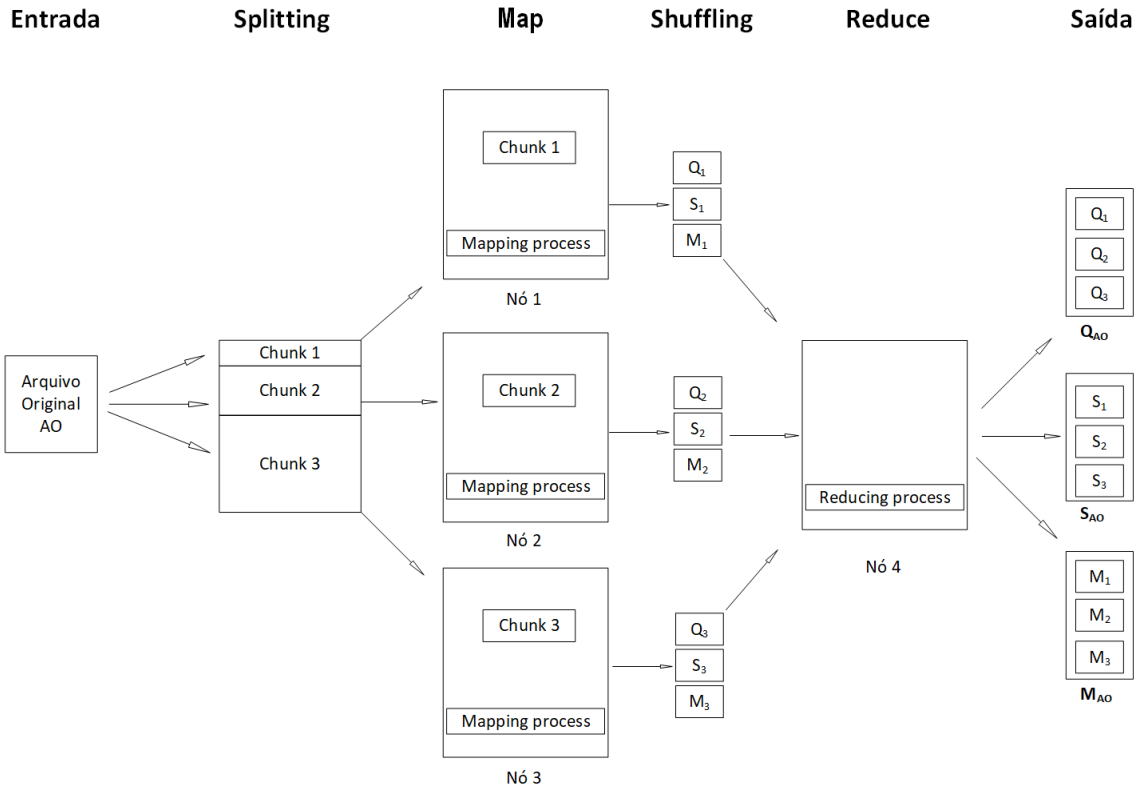

Figura 4 – Visão Geral da Estratégia pdQSM-EXTRACTION

4.1 Desafios Arquiteturais

A estratégia QSM-EXTRACTION baseia-se na fragmentação de um arquivo digital (fornecido como entrada) em fragmentos denominados objetos de informação. Mais especificamente, cada objeto de informação consiste em um bloco de 256 *b*ytes. Na abordagem centralizada adotada pelo QSM-EXTRACTION, os fragmentos (ou objetos de informação) são processados sequencialmente. Portanto, a ordem de processamento dos fragmentos já está definida, indo sempre do fragmento 1 até o fragmento k, por exemplo. Assim, para cada objeto de informação computa-se as suas características: qualidade, quantidade e medida. Em seguida, basta concatenar os dados que representam a qualidade de cada um dos objetos de informação, processados sequencialmente, para se obter a qualidade do arquivo digital fornecido como entrada. A mesma abordagem aplica-se às demais características: quantidade e medida.

Contudo, ao utilizar uma abordagem paralela e distribuída para computar as carac-

terísticas dos objetos de informação que compõem um arquivo digital, surgem alguns desafios. Primeiramente, torna-se fundamental adotar uma estratégia para identificar a ordem (sequência original) desses objetos de informação (fragmentos). Mais precisamente, é necessário identificar quem é o fragmento 1, o fragmento 2,..., o fragmento k. Essa identificação é crucial para que seja possível construir corretamente os arquivos que representam as características do arquivo digital como um todo. E, consequentemente, possibilitar a correta reconstrução do arquivo digital a partir de suas características.

Diferentemente da estratégia QSM-EXTRACTION, onde cada objeto de informação (fragmento) possui o tamanho fixo de 256 *b*ytes, o *f*ramework Spark particiona o arquivo digital fornecido como entrada em fragmentos de tamanho variável denominados *c*hunk. Assim, o tamanho de um *c*hunk é variável, sendo definido automaticamente pelo *S*park. Esses fragmentos (*c*hunks) são distribuídos e processados nos diferentes nós que compõem o *cluster*. Desta forma, o *S*park pode enviar um *c*hunk de n *b*ytes para um determinado nó “A"e outro de k *b*ytes para um nó “B", dependendo da capacidade de armazenamento e processamento de cada nó. Assim, cada nó irá processar os *c*hunks recebidos, gerando Q, S e M para cada um desses fragmentos, em uma velocidade diferente. Portanto, ao utilizar o *f*ramework Spark, é necessário identificar a sequência correta desses fragmentos (*c*hunks) que compõem o arquivo de entrada.

Para ilustrar esse problema, suponha um arquivo original de 1792 *b*ytes. Assuma ainda que o *S*park criou três *c*huncks de tamanhos 256, 512 e 1024 *b*ytes, denominados de *c*hunck 1, *c*hunck 2 e *c*hunck 3, respectivamente. Suponha também que o *c*hunck 1 foi enviado para o nó 1, o *c*hunck 2 para o nó 2 e o *c*hunck 3 para o nó 3. Considere que o nó 2 foi o primeiro a concluir o processamento a ele atribuído, gerando Q_2, S_2 e M_2 para o *c*hunck 2. Em seguida, o nó 3 concluiu sua tarefa, gerando Q_3, S_3 e M_3 para o *c*hunck 3. Por fim, o nó 1 encerrou suas atividades, produzindo Q_1, S_1 e M_1 para o *c*hunck 1. Observe que, neste caso, se nenhuma abordagem para identificação dos fragmentos (*c*huncks) for utilizada, o arquivo final gerado para armazenar as informações de Q seria uma concatenação de Q_2, Q_3 e Q_1. O mesmo ocorreria com os arquivos gerados para S e M. Consequentemente, ao se utilizarem os três arquivos contendo Q, S e M, respectivamente, para recompor o arquivo original, seria produzido um arquivo diferente do esperado. A Figura 5 ilustra um exemplo de execução que evidencia a necessidade de identificação dos fragmentos.

Adicionalmente, na estratégia QSM-EXTRACTION, tanto os vetores Q e S quanto a matriz M são variáveis globais, ou seja, podem ser acessados de qualquer ponto do código

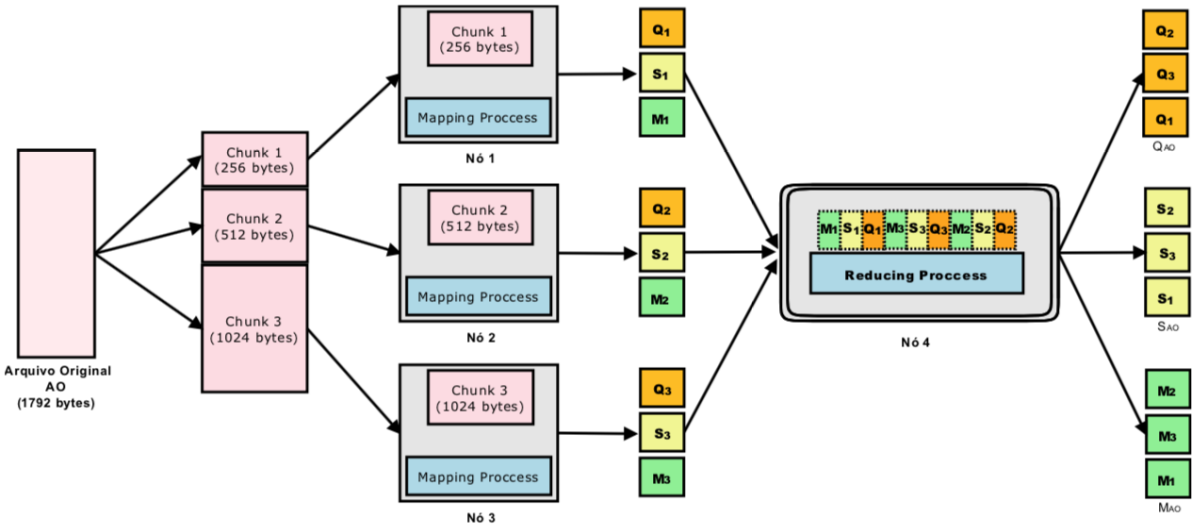

Figura 5 – Necessidade de Identificação dos Fragmentos

do programa (ou do algoritmo), uma vez que o processamento é centralizado. Porém, ao se utilizarem o paradigma MapReduce e o *f*ramework Spark, já não há variáveis (estruturas de dados) globais, uma vez que o processamento é distribuído.

Para solucionar o problema de identificar os fragmentos que compõem o arquivo digital dado como entrada, sem a utilização de estruturas globais, propomos duas abordagens distintas, as quais serão detalhadas posteriormente:

- Posição Global: Atribuir um valor sequencial para indicar a posição de cada *b*yte em *Ao*.
- Posição Local: Atribuir um valor sequencial para indicar a posição de cada *b*yte em um determinado bloco (objeto de informação) que compõe *Ao*.

4.2 Projeto e Implementação

O Algoritmo 1 descreve, de forma geral, a estratégia pdQSM-Extraction. Este algoritmo possui cinco parâmetros de entrada, os quais são descritos a seguir:

- Ao - Arquivo original;
- Tb - Tamanho do bloco (objeto de informação);
- Dq - Destino final onde será armazenado o arquivo contendo a qualidade (Q) do arquivo original (Ao);
- Ds - Destino final onde será armazenado o arquivo contendo a quantidade (S) do arquivo original (Ao);
- Dm - Destino final onde será armazenado o arquivo contendo a medida (M) do arquivo original (Ao).

Algoritmo 1: pdQSM-EXTRACTION

Input: Ao: Arquivo original, Tb: Tamanho do bloco, Dq: Destino de Q, Ds: Destino de S, Dm: Destino de M
Output: Q,S,M

1 $blocos \leftarrow$ Dividir Ao por Tb;
2 $RDD_0 \leftarrow$ Criar RDD a partir de blocos;
3 $RDD_1 \leftarrow$ Colocar índice identificador para cada entrada de RDD_0;
4 $RDDqsm \leftarrow$ Mapear RDD_1 da forma:
5 | **for *cada*** *entrada e* ***em*** RDD_1 **do**
6 | | criar tupla t $(byte, e.identificador, 1, posicao)$;
7 | | -> 1 significa uma ocorrência do byte
8 | **end**
9 $RDDqsm_O \leftarrow$Realizar ordenação em $RDDqsm$;
10 $RDDqs \leftarrow$ Mapear $RDDqsm_O$ da forma:
11 | para cada entrada e $=>$ $(e.byte, 1)$;
12 $RDDqs_r \leftarrow$ Reduzir $RDDqs$ da forma:
13 | Agrupar as tuplas com a mesma chave;
14 $RDD_Q \leftarrow$ Mapear $RDDqs_r$ da forma:
15 | para cada entrada e $=>$ $(e.byte)$;
16 Salvar RDD_Q como arquivo no destino Dq;
17 $RDD_S \leftarrow$ Mapear $RDDqs_r$ da forma:
18 | para cada entrada e $=>$ $(ocorrencia)$;
19 Salvar RDD_S como arquivo no destino Ds;
20 $RDD_M \leftarrow$ Mapear $RDDqsm_O$ da forma:
21 | para cada entrada e $=>$ $(e.posicao)$;
22 Salvar RDD_M como arquivo no destino Dm;

Inicialmente, o *f*ramework Spark define automaticamente o tamanho de cada um dos *c*huncks que serão distribuídos entre os nós. Vale ressaltar que diferentes nós podem receber *c*huncks de tamanhos distintos. Em seguida, o arquivo é particionado em um conjunto de *c*huncks *C*. Posteriormente, cada nó irá receber um determinado *c*hunk pertencente ao conjunto *C*. Vale destacar que um *c*hunck é um fragmento físico, ou seja, que será fisicamente enviado e armazenado em um determinado nó do *cluster*.

Na linha 1 do Algoritmo 1 é criado um conjunto de *n* blocos a partir da fragmentação do arquivo original Ao, onde *n* é igual ao tamanho de Ao dividido por Tb. Vale ressaltar que esses fragmentos constituem blocos lógicos. Assim, um *chunck*, fragmento físico, pode conter um ou mais blocos lógicos. Cada bloco lógico representa um objeto de informação.

Posteriormente, na linha 2 cria-se um RDD inicial, chamado RDD_0, com base no conjunto de blocos criado anteriormente. O RDD_0 é formado por um conjunto de entradas (linhas), onde cada linha armazena um dos blocos anteriormente criados. Logo, cada entrada do RDD_0 consiste em um array de *b*ytes de tamanho Tb. Vale ressaltar que essa operação é

realizada de forma distribuída, onde cada nó opera sobre o *c*hunk armazenado localmente, em geral em memória. O RDD é um conceito centralizado, mas fisicamente ele está distribuído entre os diversos nós. Assim, cada nó armazena um fragmento do RDD, montado de acordo com os *c*hunks armazenados localmente. A Figura 6 ilustra o relacionamento entre o RDD_0 e os *c*hunks criados pelo *S*park.

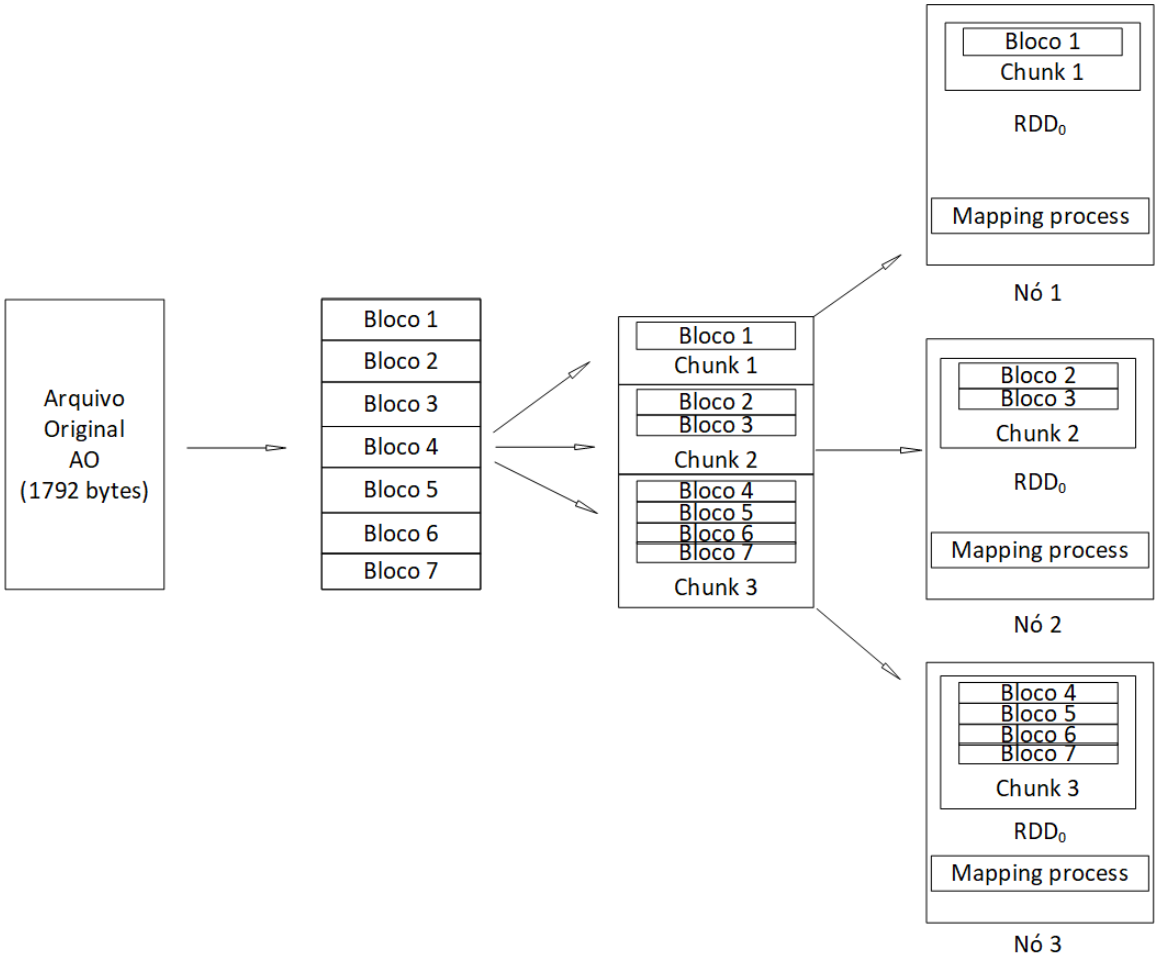

Figura 6 – Relacionamento entre o RDD_0 e os *Chunks*

Um novo RDD, chamado RDD_1, é criado na linha 3, a partir do RDD_0, adicionando-se um índice (identificador) a cada entrada. Desta forma, o RDD_1 é um conjunto de entradas, onde cada entrada *e* é uma tupla contendo dois valores: o identificador (*e*.identificador) e o array de *b*ytes de tamanho Tb (*e*.bloco). A Figura 7 ilustra a estrutura do RDD_1, enquanto a Figura 8 mostra um possível exemplo do RDD_1 e a Figura 9 ilustra o relacionamento entre o RDD_1 e os *Chunks* criados pelo *S*park.

As linhas de 4 a 8 ilustram a execução da função Map, a qual retorna um novo RDD chamado RDD_{qsm}, gerado da seguinte forma. Percorre-se cada entrada *e* de RDD_1, onde *e* é

Figura 7 – Ilustração do RDD_1

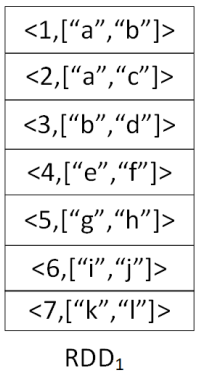

Figura 8 – Exemplo do RDD_1

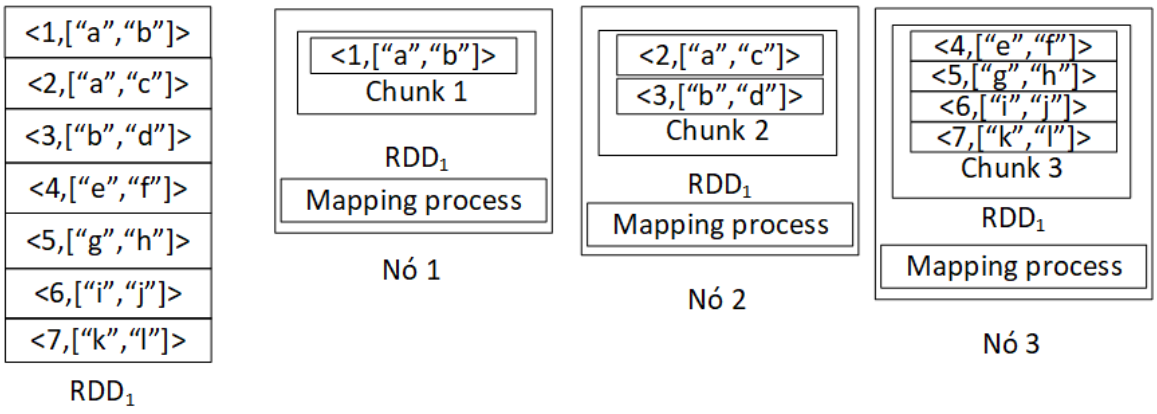

Figura 9 – Relacionamento entre o RDD_1 e os *Chunks*

uma tupla do tipo <e.identificador, e.bloco> (linha 5). Em seguida, dada uma determinada entrada *e*, para cada *byte* *b* em e.bloco cria-se uma tupla *t* do tipo < *b*, e.identificador, 1, posição de *b* em Ao>. O valor 1 na terceira posição da tupla *t* indica que uma ocorrência do *byte* *b* foi encontrada. Serão utilizadas duas abordagens distintas para representar a posição de *b* em Ao. Essas abordagens serão discutidas em profundidade nas próximas seções. Por fim, a tupla *t* é adicionada em RDD_{qsm}. A função Map é executada de forma distribuída e paralela, onde cada nó executa essa tarefa para as entradas que pertencem ao fragmento do RDD armazenado

localmente. A Figura 10 ilustra um possível exemplo do RDD_{qsm}. Já a Figura 11 mostra o relacionamento entre o RDD_{qsm} e os *c*hunks criados pelo *S*park.

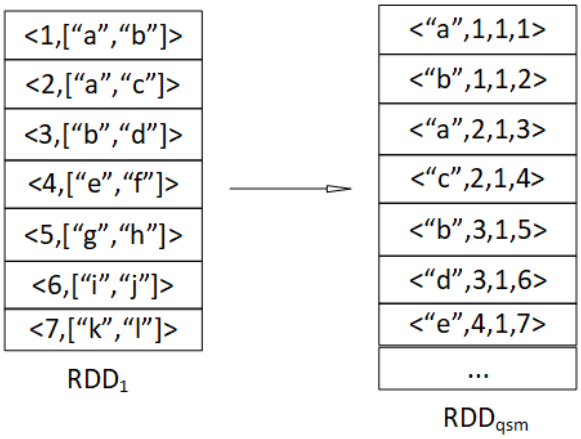

Figura 10 – Ilustração do RDD_{qsm}

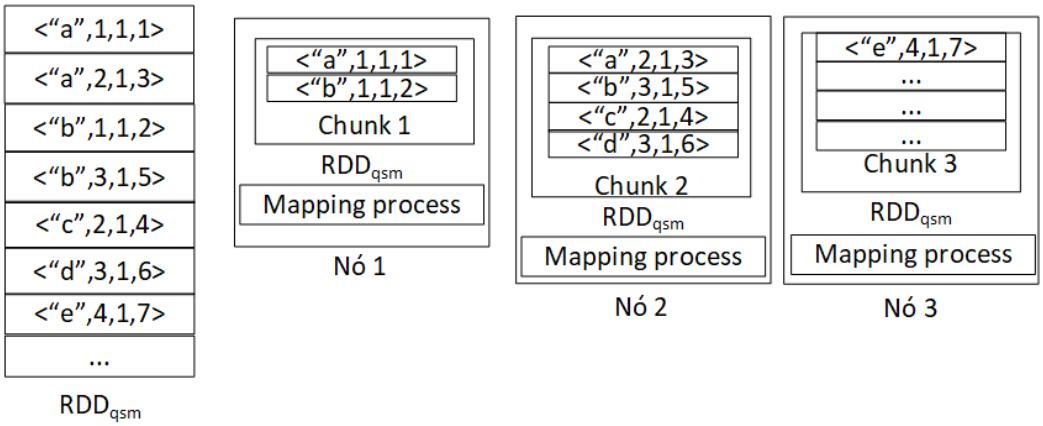

Figura 11 – Relacionamento entre o RDD_{qsm} e os *Chunks*

Na linha 9, executa-se uma operação de ordenação sobre as entradas de RDD_{qsm}, dando origem a um novo RDD chamado RDD_{qsmo}. Serão utilizadas duas abordagens distintas para essa ordenação: i) ordenação pelo *b*yte *b* e ii) ordenação pelo identificador da entrada (e.identificador). Essas abordagens serão discutidas em detalhes nas próximas seções. Em ambos os casos, cada nó irá ordenar os dados armazenados localmente, de forma distribuída. A Figura 12 ilustra um possível exemplo do RDD_{qsmo} utilizando ordenação pelo *b*yte *b*. Já a Figura 13 ilustra um possível exemplo do RDD_{qsmo} utilizando ordenação pelo identificador da entrada (e.identificador).

Nas linhas 10 e 11, executa-se uma segunda operação Map, tendo como entrada RDD_{qsmo}, o qual possui um conjunto de entradas ordenadas do tipo $< b$, e.identificador, 1, posição de *b* em Ao$>$, e como saída um novo RDD chamado RDD_{qs}. Esta operação é realizada

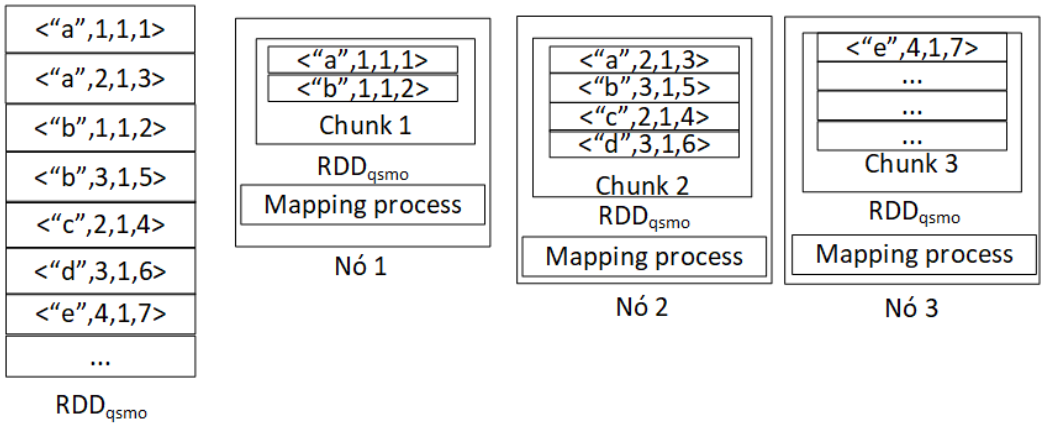

Figura 12 – RDD_{qsmo} utilizando ordenação pelo *b*yte *b*

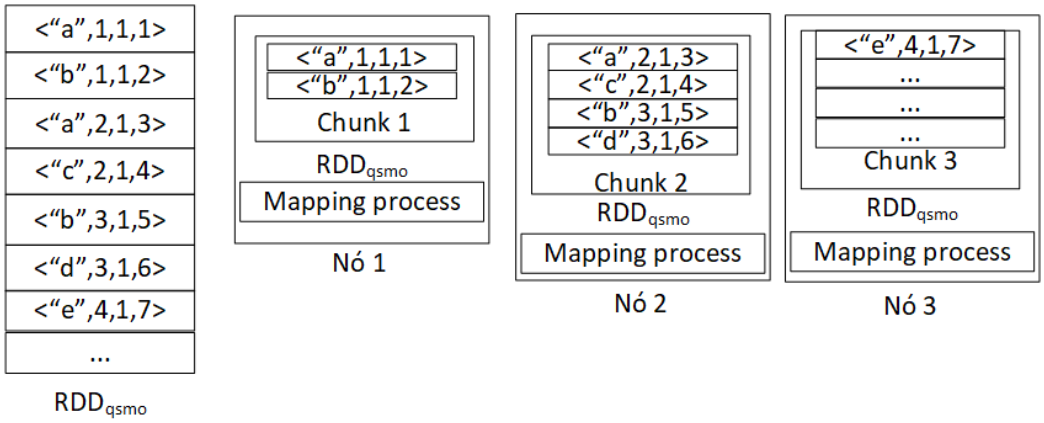

Figura 13 – RDD_{qsmo} utilizando ordenação pelo identificador da entrada (e.identificador)

da seguinte forma. Para cada entrada *e* em RDD_{qsmo}, cria-se uma tupla *t* da forma $< b, 1>$. Em seguida, adiciona-se a tupla *t* em RDD_{qs}. A Figura 14 ilustra um possível exemplo do RDD_{qs} utilizando ordenação pelo *b*yte *b*.

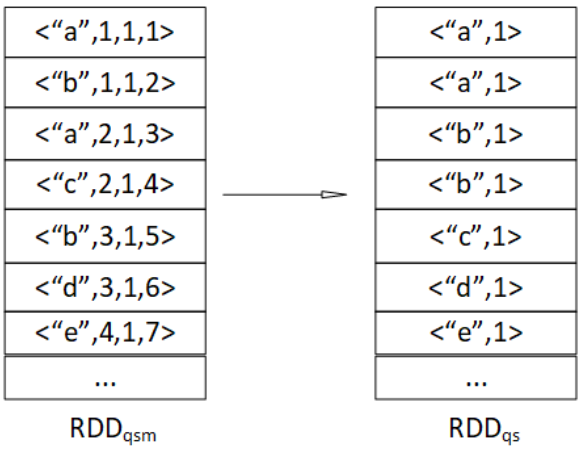

Figura 14 – RDD_{qs} utilizando ordenação pelo *b*yte *b*

Nas linhas 12 e 13, executa-se uma operação Reduce tendo como entrada RDD_{qs} (contendo entradas no formato $< b, 1>$) e como saída um novo RDD chamado RDD_{qs_r}. Esta

operação é realizada da seguinte forma. Agrupam-se as entradas de RDD_{qs} com a mesma chave (*b*yte *b*) e para cada grupo cria-se uma nova tupla *t* do tipo <chave, valor> contendo como chave o *b*yte *b* e como valor o somatório da coluna valor de todas as tuplas do grupo. Desta forma, o RDD RDD_{qs_r} representa a quantidade e a qualidade de Ao, uma vez que suas entradas são do tipo $<$*b*yte *b*, quantidade de ocorrências de $b>$. A Figura 15 ilustra um possível exemplo do RDD_{qs_r}.

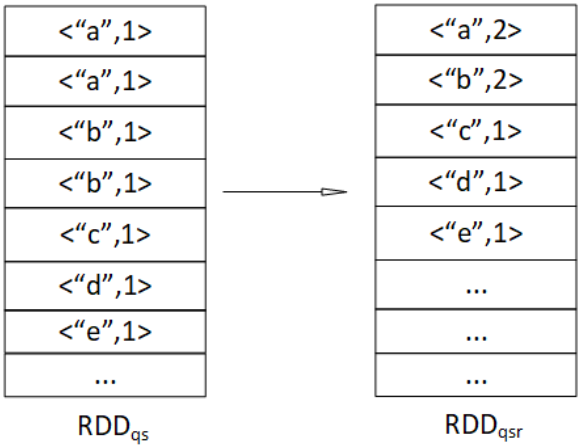

Figura 15 – Um Possível Exemplo de RDD_{qs_r}

Nas linhas 14 e 15, aplica-se uma função Map tendo como entrada RDD_{qs_r} e como saída um novo RDD chamado RDD_Q. Esta operação é executada da seguinte forma. Para cada entrada *e* em RDD_{qs_r} é gerada uma nova tupla *t* da forma $< b >$. Em seguida, adiciona-se a tupla *t* em RDD_Q. Por fim, o RDD_Q é armazenado em *Dq* (linha 16). A Figura 16 ilustra um possível exemplo do RDD_Q.

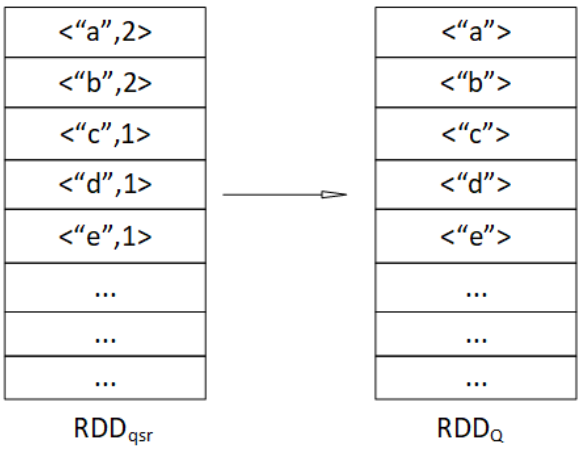

Figura 16 – Um Possível Exemplo de RDD_Q

Nas linhas 17 e 18, aplica-se uma função Map tendo como entrada RDD_{qs_r} e como

saída um novo RDD chamado RDD_S. Esta operação é executada da seguinte forma. Para cada entrada e em RDD_{qs_r} é gerada uma nova tupla t da forma <quantidade de ocorrências de b >. Em seguida, adiciona-se a tupla t em RDD_S. Por fim, o RDD_S é armazenado em Ds (linha 19). A Figura 17 ilustra um possível exemplo do RDD_S.

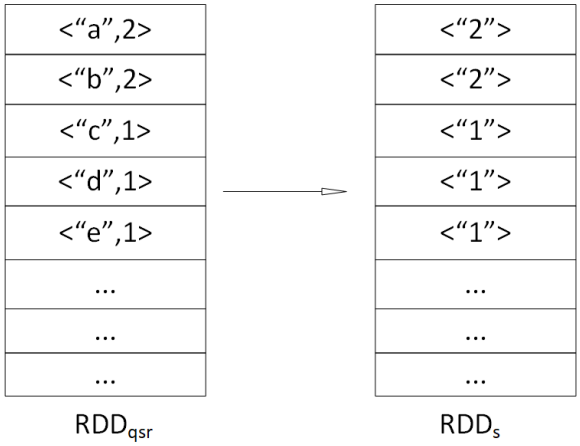

Figura 17 – Um Possível Exemplo de RDD_S

Nas linhas 20 e 21, aplica-se uma função Map tendo como entrada RDD_{qsmo} e como saída um novo RDD chamado RDD_M. Vale lembrar que as entradas em RDD_{qsmo} possuem o seguinte formato: $<$ b, e.identificador, 1, posição de b em Ao$>$. Esta operação Map é executada da seguinte forma. Para cada entrada e em RDD_{qsmo} é gerada uma nova tupla t da forma <posição de b em Ao$>$. Em seguida, adiciona-se a tupla t em RDD_M. Por fim, o RDD_M é armazenado em Dm (linha 22). A Figura 18 ilustra um possível exemplo do RDD_M.

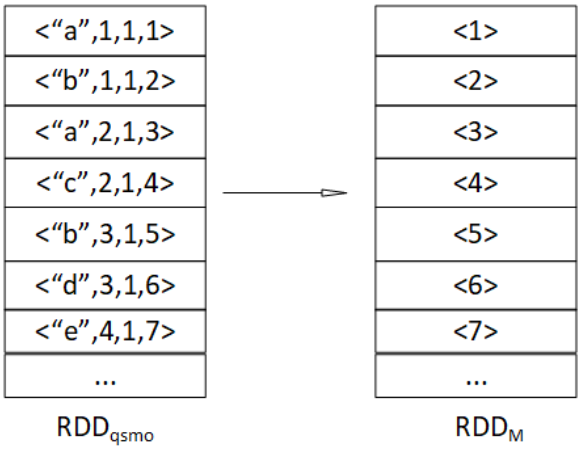

Figura 18 – Um Possível Exemplo de RDD_M

4.2.1 Abordagem 1: Posição Global

Conforme mencionado anteriormente, ao utilizar-se o paralelismo presente no *f*ramework Spark, torna-se necessário identificar a sequência correta dos objetos de informação que compõem o arquivo de entrada. A abordagem denominada Posição Global utiliza a seguinte estratégia para identificar essa sequencia. Suponha que a variável de entrada Tb (Tamanho do Bloco) tenha sido definida com o valor 256, ou seja, cada bloco possui 256 *bytes*. Considere um arquivo digital *Ao* fornecido como entrada contendo 128.000 *bytes*. Neste caso, o arquivo *Ao* será fragmentado em 500 blocos. Estes blocos são numerado sequencialmente, ou seja, de 0 a 499. Já os *bytes* que compõem cada um desses blocos também são numerados sequencialmente de 0 a 127.999. Assim, mesmo que os blocos sejam processados de forma distribuída e paralela, é possível, ao final deste processamento, ordenar os blocos e seus respectivos *bytes* com a finalidade de representar corretamente as características do arquivo de entrada. Nesta primeira implementação da estratégia proposta, o parâmetro de entrada Tb (Tamanho do bloco, ou seja, do objeto de informação) foi inicializado com o valor 256. Este valor foi escolhido com o objetivo de possibilitar uma comparação mais justa com a estratégia QSM-EXTRATION, a qual constitui o nosso *baseline*.

O Algoritmo 2 descreve a estratégia pdQSM-EXTRACTION utilizando a abordagem de Posição Global. Vale destacar que o Algoritmo 2 possui apenas duas diferenças em relação ao Algoritmo 1, as quais serão destacadas a seguir. A primeira alteração ocorre na Linha 9 do O Algoritmo 2, onde executa-se uma operação de ordenação, pelo *b*yte, sobre as entradas de RDD_{qsm}, dando origem a um novo RDD chamado RDD_{qsmo}, conforme ilustrado na Figura 12. A segunda diferença ocorre na Linha 22, onde o RDD_M passa por um processo de compressão GZip, utilizando-se uma funcionalidade do próprio *Spark*, antes de ser armazenado em *Dm*. Esta compressão foi utilizada para tentar diminuir o tamanho do arquivo contendo a Medida de *Ao*.

4.2.2 Abordagem 2: Posição Local

A abordagem denominada Posição Local utiliza a seguinte estratégia para identificar a sequencia correta dos objetos de informação que compõem o arquivo de entrada. Suponha que a variável de entrada Tb (Tamanho do Bloco) tenha sido definida com o valor 256, ou seja, cada bloco possui 256 *bytes*. Considere um arquivo digital *Ao* fornecido como entrada contendo 128.000 *bytes*. Neste caso, o arquivo *Ao* será fragmentado em 500 blocos. Estes

Algoritmo 2: pdQSM-EXTRACTION Utilizando Posição Global

Input: *Ao*: Arquivo original, Tb: Tamanho do bloco, *Dq*: Destino de Q, *Ds*: Destino de S, *Dm*: Destino de M
Output: *Q*,*S*,*M*

1 $blocos \leftarrow$ Dividir *Ao* por *Tb*;
2 $RDD_0 \leftarrow$ Criar RDD a partir de blocos;
3 $RDD_1 \leftarrow$ Colocar índice identificador para cada entrada de RDD_0;
4 *RDDqsm* $\leftarrow$ Mapear RDD_1 da forma:
5 | **for *cada*** *entrada e* ***em*** RDD_1 **do**
6 | | criar tupla t $(byte, e.identificador, 1, posicao)$;
7 | | -> 1 significa uma ocorrência do byte
8 | **end**
9 $RDDqsm_O \leftarrow$Realizar ordenação em *RDDqsm* pelo byte;
10 *RDDqs* $\leftarrow$ Mapear $RDDqsm_O$ da forma:
11 | para cada entrada $e => (e.byte, e.ocorrencia)$;
12 $RDDqs_r \leftarrow$ Reduzir *RDDqs* da forma:
13 | Agrupar as tuplas com a mesma chave;
14 $RDD_Q \leftarrow$ Mapear $RDDqs_r$ da forma:
15 | para cada entrada $e => (e.byte)$;
16 Salvar RDD_Q como arquivo no destino *Dq*;
17 $RDD_S \leftarrow$ Mapear $RDDqs_r$ da forma:
18 | para cada entrada $e => (e.ocorrencia)$;
19 Salvar RDD_S como arquivo no destino *Ds*;
20 $RDD_M \leftarrow$ Mapear $RDDqsm_O$ da forma:
21 | para cada entrada $e => (e.posicao)$;
22 Salvar RDD_M como arquivo no destino *Dm* aplicando compressão Gzip;

blocos são numerado sequencialmente, ou seja, de 0 a 499. Já os *bytes* que compõem cada um desses blocos serão numerados de 0 a 255. Assim, mesmo que os blocos sejam processados de forma distribuída e paralela, é possível, ao final deste processamento, ordenar os blocos e seus respectivos *bytes* com a finalidade de representar corretamente as características do arquivo de entrada. Nesta segunda implementação da estratégia proposta, o parâmetro de entrada Tb (Tamanho do bloco, ou seja, do objeto de informação) foi inicializado com o valor 256. Este valor foi escolhido com o objetivo de possibilitar uma comparação mais justa com a estratégia QSM-EXTRATION, a qual constitui o nosso *baseline*.

O Algoritmo 3 descreve a estratégia pdQSM-EXTRACTION utilizando a abordagem de Posição Local. A seguir, as diferenças entre o Algoritmo 3 (Posição Local) e o Algoritmo 2 (Posição Global).

A primeira diferença encontra-se na linha 9 do Algoritmo 3. Nesta linha, executa-se uma operação de ordenação, pelo identificador da entrada (e.identificador), sobre as entradas de RDD_{qsm}, dando origem a um novo RDD chamado RDD_{qsmo}. A Figura 19 ilustra um possível

Algoritmo 3: pdQSM-EXTRACTION Utilizando Posição Local

Input: *Ao*: Arquivo original, Tb: Tamanho do bloco, *Dq*: Destino de Q, *Ds*: Destino de S, *Dm*: Destino de M
Output: *Q*,*S*,*M*

1 $blocos \leftarrow$ Dividir *Ao* por *Tb*;
2 $RDD_0 \leftarrow$ Criar RDD a partir de blocos;
3 $RDD_1 \leftarrow$ Colocar índice identificador para cada entrada de RDD_0;
4 *RDDqsm* $\leftarrow$ Mapear RDD_1 da forma:
5 | **for *cada*** *entrada e* ***em*** RDD_1 **do**
6 | | criar tupla t $(byte, e.identificador, 1, posicao)$;
7 | | -> 1 significa uma ocorrência do byte
8 | **end**
9 $RDDqsm_O \leftarrow$Realizar ordenação em *RDDqsm* pelo identificador da entrada;
10 *RDDmapkeys* $\leftarrow$ Mapear $RDDqsm_O$ da forma:
11 | para cada entrada e $=>$ $((e.identificador, e.byte), (e.ocorrencia, e.posicao))$;
12 *RDDqs* $\leftarrow$ Mapear $RDD_{mapkeys}$ da forma:
13 | para cada entrada e $=>$ $((e.identificador, e.byte), e.ocorrencia)$;
14 $RDDqs_r \leftarrow$ Reduzir *RDDqs* da forma:
15 | Agrupar as tuplas com a mesma chave;
16 $RDD_Q \leftarrow$ Mapear $RDDqs_r$ da forma:
17 | para cada entrada $e =>$ $(e.byte)$;
18 Salvar RDD_Q como arquivo no destino *Dq*;
19 $RDD_S \leftarrow$ Mapear $RDDqs_r$ da forma:
20 | para cada entrada $e =>$ $(e.ocorrencia)$;
21 Salvar RDD_S como arquivo no destino *Ds*;
22 *RDDmo* $\leftarrow$ Mapear $RDD_{mapkeys}$ da forma:
23 | para cada entrada $e =>$ $((e.identificador, e.byte), e.posicao)$;
24 *RDDmagg* $\leftarrow$ Agregar *RDDmo* da seguinte forma:
25 | para cada entrada *e* de mesmo e.byte $=>$ $e.pocicao \leftarrow$ ($posicao_1$ + ","+ $posicao_2$ + ","+ ..... + ","+ $posicao_N$);
26 $RDD_M \leftarrow$ Mapear *RDDmagg* da forma:
27 | para cada entrada $e =>$ $(e.posicao)$;
28 Salvar RDD_M como arquivo no destino *Dm* aplicando compressão Gzip;

exemplo do RDD_{qsmo}.

Nas linhas 10 e 11, executa-se uma segunda operação Map, tendo como entrada RDD_{qsmo}, o qual possui um conjunto de entradas ordenadas do tipo $< b$, e.identificador, 1, posição de *b* em Ao$>$, e como saída um novo RDD chamado $RDD_{mapkeys}$. Esta operação é realizada da seguinte forma. Para cada entrada *e* em RDD_{qsmo}, cria-se uma tupla *t* da forma $<$(e.identificador, *e.byte*), (e.ocorrência, e.posição)$>$. Em seguida, adiciona-se a tupla *t* em $RDD_{mapkeys}$. A Figura 20 ilustra um possível exemplo do $RDD_{mapkeys}$.

Nas linhas 12 e 13, executa-se uma segunda operação Map, tendo como entrada $RDD_{mapkeys}$, o qual possui um conjunto de entradas ordenadas do tipo $<$(e.identificador, *e.byte*),

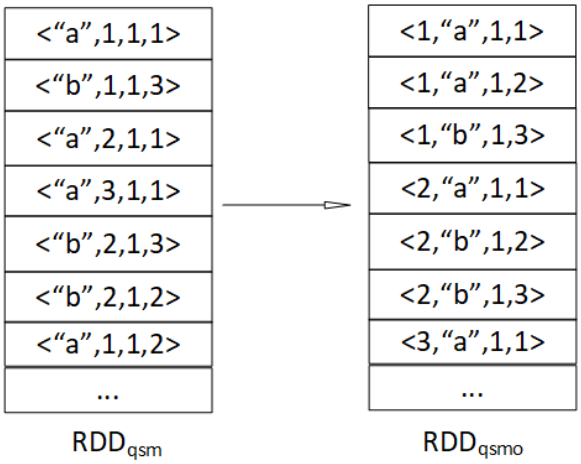

Figura 19 – Um Possível Exemplo de RDD_{qsmo} do algoritmo 3

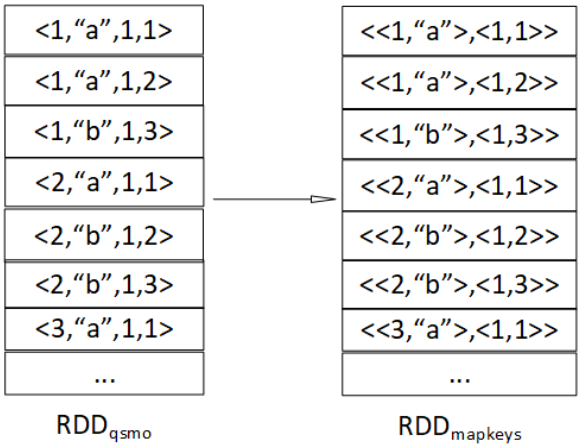

Figura 20 – Um Possível Exemplo de $RDD_{mapkeys}$ no Algoritmo 3

(e.ocorrência, e.posição)>, e como saída um novo RDD chamado RDD_{qs}. Esta operação é realizada da seguinte forma. Para cada entrada e em $RDD_{mapkeys}$, cria-se uma tupla t da forma <(*e.identificador*,*e.byte*), (e.ocorrência)>. Em seguida, adiciona-se a tupla t em RDD_{qs}. A Figura 21 ilustra um possível exemplo do RDD_{qs}, o qual representa a quantidade e a qualidade de *Ao*.

Nas linhas 22 e 23, executa-se uma segunda operação Map, tendo como entrada $RDD_{mapkeys}$, o qual possui um conjunto de entradas ordenadas do tipo <(e.identificador, *e.byte*), (e.ocorrência, e.posição)>, e como saída um novo RDD chamado RDD_{mo}. Esta operação é realizada da seguinte forma. Para cada entrada e em $RDD_{mapkeys}$, cria-se uma tupla t da forma <(e.identificador,*e.byte*), e.posição>. Em seguida, adiciona-se a tupla t em RDD_{mo}. A Figura 22 ilustra um possível exemplo do RDD_{mo}.

Nas linhas 24 e 25, executa-se uma segunda operação de agregação, tendo como entrada RDD_{mo}, o qual possui um conjunto de entradas ordenadas do tipo <(e.identificador,*e.byte*),

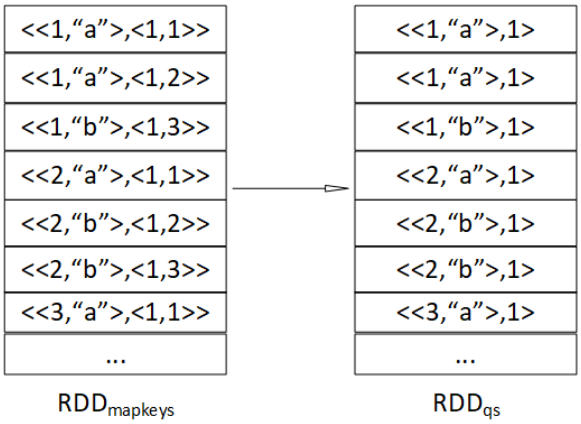

Figura 21 – Um Possível Exemplo de RDD_{qs} no Algoritmo 3

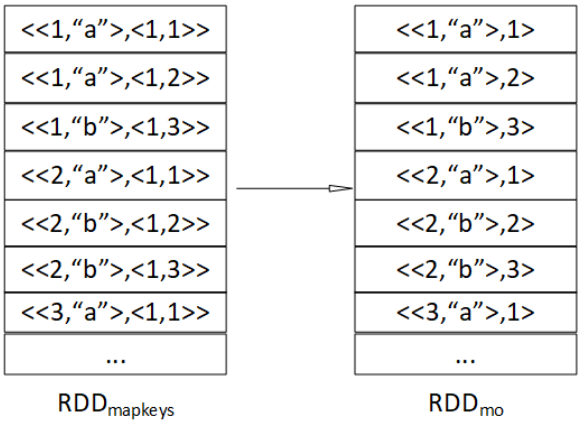

Figura 22 – Um Possível Exemplo de RDD_{mo} no Algoritmo 3

e.posição>, e como saída um novo RDD chamado RDD_{magg}. Esta operação é realizada da seguinte forma. Agrupa-se todas as entradas que possuem a mesma chave (e.identificador,*e.byte*), cria-se uma tupla t da forma $<$(e.identificador,*e.byte*), (e.posição$_1$, posição$_2$, $\cdots$ posição$_n$) $>$. Em seguida, adiciona-se a tupla t em RDD_{magg}. A Figura 23 ilustra um possível exemplo do $RDD_{rddmagg}$.

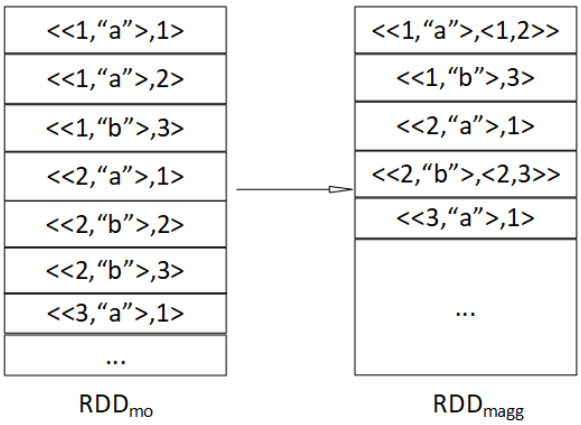

Figura 23 – Um Possível Exemplo de RDD_{magg} no Algoritmo 3

Nas linhas 26 e 27, aplica-se uma função Map tendo como entrada RDD_{magg} e como saída um novo RDD chamado RDD_M. Vale lembrar que as entradas em RDD_{magg} possuem o seguinte formato: $<$(e.identificador,*e.byte*), (e.posição$_1$, posição$_2$, $\cdots$ posição$_n$)$>$. Esta operação Map é executada da seguinte forma. Para cada entrada *e* em RDD_{magg} é gerada uma nova tupla *t* da forma $<$e.posição$_1$, posição$_2$, $\cdots$ posição$_n>$. Posteriormente, adiciona-se a tupla *t* em RDD_M. Por fim, na Linha 28, o RDD_M é submetido a um processo de compressão GZip, utilizando-se uma funcionalidade do próprio *Spark*, e em seguida é armazenado em *Dm*. A Figura 24 ilustra um possível exemplo do RDD_M.

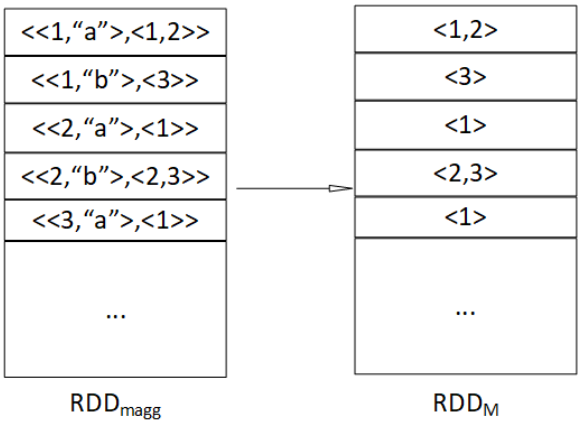

Figura 24 – Um Possível Exemplo de RDD_M no Algoritmo 3

4.2.3 Abordagem 3: Posição Local e Compressão

O Algoritmo 4 descreve a terceira versão (implementação) da estratégia pdQSM-EXTRACTION. Vale destacar que o Algoritmo 4 possui apenas duas diferenças em relação ao Algoritmo 3, as quais serão destacadas a seguir. A primeira alteração ocorre na Linha 18 do O Algoritmo 4, onde o RDD_Q passa por um processo de compressão GZip, utilizando-se uma funcionalidade do próprio *Spark*, antes de ser armazenado em *Dq*. A segunda diferença ocorre na Linha 21, onde o RDD_S passa por um processo de compressão GZip, utilizando-se uma funcionalidade do próprio *Spark*, antes de ser armazenado em *Ds*. Estas duas compressões foram utilizada para tentar diminuir o tamanho dos arquivos contendo a Qualidade e a Quantidade de *Ao*. A utilização da compressão nesses dois arquivos mostrou-se bastante interessante uma vez que estes possuem um número elevado de valores repetidos.

Algoritmo 4: pdQSM-EXTRACTION Utilizando Posição Local e Compressão

Input: *Ao*: Arquivo original, Tb: Tamanho do bloco, *Dq*: Destino de Q, *Ds*: Destino de S, *Dm*: Destino de M
Output: *Q*,*S*,*M*

1 $blocos \leftarrow$ Dividir *Ao* por *Tb*;
2 $RDD_0 \leftarrow$ Criar RDD a partir de blocos;
3 $RDD_1 \leftarrow$ Colocar índice identificador para cada entrada de RDD_0;
4 *RDDqsm* $\leftarrow$ Mapear RDD_1 da forma:
5 | **for** ***cada*** *entrada e* ***em*** RDD_1 **do**
6 | | criar tupla t $(byte, e.identificador, 1, posicao)$;
7 | | -> 1 significa uma ocorrência do byte
8 | **end**
9 $RDDqsm_O \leftarrow$Realizar ordenação em *RDDqsm* pelo identificador de entrada;
10 *RDDmapkeys* $\leftarrow$ Mapear $RDDqsm_O$ da forma:
11 | para cada entrada e $=>$ $((e.identificador, e.byte), (e.ocorrencia, e.posicao))$;
12 *RDDqs* $\leftarrow$ Mapear *RDDmapkeys* da forma:
13 | para cada entrada e $=>$ $((e.identificador, e.byte), e.ocorrencia)$;
14 $RDDqs_r \leftarrow$ Reduzir *RDDqs* da forma:
15 | Agrupar as tuplas com a mesma chave;
16 $RDD_Q \leftarrow$ Mapear $RDDqs_r$ da forma:
17 | para cada entrada $e =>$ $(e.byte)$;
18 Salvar RDD_Q como arquivo no destino *Dq* aplicando compressão Gzip;
19 $RDD_S \leftarrow$ Mapear $RDDqs_r$ da forma:
20 | para cada entrada $e =>$ $(e.ocorrencia)$;
21 Salvar RDD_S como arquivo no destino *Ds* aplicando compressão Gzip;
22 *RDDmo* $\leftarrow$ Mapear $RDD_{mapkeys}$ da forma:
23 | para cada entrada $e =>$ $((e.identificador, e.byte), e.posicao)$;
24 *RDDmagg* $\leftarrow$ Agregar *RDDmo* da seguinte forma:
25 | para cada entrada *e* de mesmo e.byte $=>$ $e.pocicao \leftarrow (posicao_1$ + ","+ $posicao_2$ + ","+ ..... + ","+ $posicao_N)$;
26 $RDD_M \leftarrow$ Mapear *RDDmagg* da forma:
27 | para cada entrada $e =>$ $(e.posicao)$;
28 Salvar RDD_M como arquivo no destino *Dm* aplicando compressão Gzip;

4.2.4 Recomposição

Apesar do presente trabalho ter por finalidade paralelizar apenas a fase de decomposição da estratégia QSM-EXTRACTION, uma versão do algoritmo de recomposição foi implementado em linguagem Scala, de forma centralizada, com a finalidade de avaliar a corretude da estratégia pdQSM-EXTRACTION. Mais precisamente, o objetivo é permitir recompor o arquivo original a partir dos arquivos gerados pela estratégia pdQSM-EXTRACTION. O Algoritmo 5 descreve a recomposição utilizada pela pdQSM-EXTRACTION. Como entrada do algoritmo temos os arquivos de Qualidade (Q), Quantidade (S) e Medida (M), além do destino

(D) onde o arquivo reconstruído será salvo. Inicialmente, o algoritmo cria três *arrays*:

- ***Qualidade*** - Armazena valores de **Q**, referentes aos *bytes* presentes no Arquivo Original;
- ***Quantidade*** - Armazena os valores de **S**, referentes ao número de vezes que cada *byte* ocorre no Arquivo Original;
- ***Medida*** - Armazena os valores de **M**, referentes às posições em que cada *byte* ocorre no Arquivo Original;

Em seguida, calculamos o tamanho do Arquivo Original a partir do maior valor de *Medida*, denominado *TamanhoAo*. Para representar a posição do *array Medida* que está sendo computada no momento da execução do algoritmo criamos uma variável chamada *linhaM*, inicializada com valor 0, representando a primeira posição da *Medida*.

O arquivo original é representado pelo *array Ao*, do tipo *Byte* e de tamanho *TamanhoAo*. Na linha 7 do algoritmo, iniciamos a reconstrução de *Ao*. Para cada *byte b* no *array* Qualidade, iniciamos um laço com uma variável *k*, a qual varia de *linhaM*(inicialmente 0), indicando a primeira posição do *byte b* no *array Medida*, até a quantidade de vezes em que o *byte b* ocorre no Arquivo Original. Dessa forma, temos que armazenar esse *byte* nas posições *Medida(k)*, que são as posições em que o *byte* apareceu no Arquivo Original, conforme podemos observar na linha 9.

Encerrado o laço interior, atualizamos o valor de *linhaM*, na linha 11, para *linhaM* + *Quantidade(b)*, preparando para a execução do próximo *byte*. Finalizamos salvando o Arquivo Original *Ao* no destino *D*.

Algoritmo 5: Recomposição

```
   Input: Q: Qualidade, S: Quantidade, M: Medida, D: Destino do Arquivo
   Output: Ao: Arquivo original
 1 Qualidade ← Criar array apartir de Q ;
 2 Quantidade ← Criar array a partir de S;
 3 Medida ← Criar array a partir de M;
 4 TamanhoAo ← Maior valor de Medida;
 5 linhaM ← 0                  -> Variável auxiliar para indicar a posição atual em Medida;
 6 Ao ←Array[Byte](TamanhoAo)      -> Array de tamanho TamanhoAo e do tipo Byte;
 7 for cada byte b em Qualidade do
 8 |   for k de linhaM até (Quantidade(b) + (linhaM -1)) do
 9 |   |   Ao(Medida(k)) ← byte b      -> Ao recebe o byte b na posição original
10 |   end
11 |   linhaM ← linhaM + Quantidade(b)      -> Atualiza posição atual ;
12 end
13 Salvar arquivo Ao no destino D;
```

4.2.5 Implementação

As implementações dos algoritmos descritos neste capítulo, utilizando-se a linguagem Scala e a plataforma *Spark* podem ser encontradas no repositório GitHub disponibilizado a seguir: https://github.com/lucasmdantas/pdQSM-Extraction.

5 AVALIAÇÃO EXPERIMENTAL

5.1 Introdução

Neste capítulo são apresentados os experimentos realizados com a finalidade de avaliar o desempenho da estratégia pdQSM-EXTRACTION. Para executar estes experimentos, foi utilizada uma infraestrutura de nuvem pública gerida pelo Amazon. A Figura 25 ilustra a arquitetura utilizada nos experimentos, a qual contém quatro máquinas virtuais, sendo um nó *Master* e três nós Escravos. O nó *Master* realiza a função de Terceira Parte Confiável (TPC) atendendo as requisições de armazenamento e recuperação de arquivos enviadas pelo cliente (usuários e/ou aplicações). Adicionalmente, o nó *Master* executa as funções de orquestração do *framework* SPARK. Os nós Escravos executam as funções *Map* e *Reduce*. Cada nó utilizado tinha a seguinte configuração: Sistema Operacional Ubuntu 14.04, processador Intel(R) Xeon(R) CPU 2.40GHz, 4 núcleos, 16GB de RAM e 90GB de armazenamento SSD. Em cada um dos nós escravos, um core foi deixado livre e os outros três cores foram configurados como executores. Todas essas quatro máquinas virtuais estavam sob a gerência do SPARK.

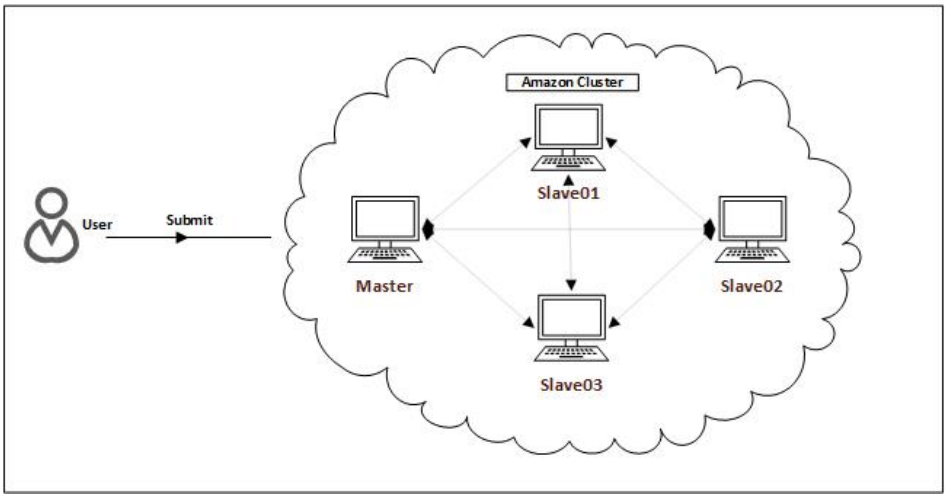

Figura 25 – Arquitetura Utilizada nos Experimentos

Neste ambiente de execução, um cliente que necessita armazenar um arquivo **F**, envia uma requisição para o nó *Master* (TPC) armazenar um arquivo nas nuvens, juntamente com o arquivo a ser armazenado. Em seguida, o nó *Master*, por meio do *framework* SPARK, distribui entre os executores, presentes nos nós escravos, as tarefas de *Map* e *Reduce*, as quais possuem a finalidade de extrai as características de Qualidade, Quantidade e Medida dos objetos de informação que compõem o arquivo do cliente e as armazena em três destinos fisicamente

distintos. Vale destacar que, para avaliar o algoritmo QSM-EXTRACTION, uma vez que este é um algoritmo centralizado, utilizamos o nó *Master*, como TPC, e um nó escravo, o qual foi responsável por extrair as características de Qualidade, Quantidade e Medida dos objetos de informação que compõem o arquivo do cliente. O processo de recuperação do arquivo por parte do cliente não foi avaliado, uma vez que está fora do escopo desta dissertação.

Os algoritmos que compõem a estratégia QSM-EXTRACTION foram implementados em linguagem C++. Já os algoritmos que compõem a estratégia pdQSM-EXTRACTION foram implementados em linguagem Scala.

5.2 Resultados Experimentais

Com a finalidade de avaliar a eficiência da estratégia pdQSM-EXTRACTION, utilizamos como métrica o tempo de entrada, o qual é definido como sendo o tempo total gasto para decompor um arquivo **F** e gerar os 3 arquivos contendo as características de qualidade (Fq.bin), quantidade (Fs.bin) e medida (Fm.bin) de **F**. Esses 3 arquivos serão enviados para os provedores de serviços de armazenamento na nuvem. É importante destacar que o tempo gasto para recompor o arquivo de entrada **F**, a partir dos três arquivos que contêm suas características (Fq.bin, Fs.bin e Fm.bin), bem como o tempo gasto no processo de comunicação, ou seja, para enviar e receber dados entre o cliente e os provedores de nuvem, não foram avaliados. Foram realizados testes com diferentes tipos e tamanhos de arquivos. Para cada tamanho de arquivo distinto, o processo foi repetido 10 vezes.

A estratégia pdQSM-EXTRACTION foi avaliada a partir de quatro cenários distintos, que serão apresentados a seguir. O primeiro cenário teve por finalidade analisar o desempenho das estratégias QSM-EXTRACTION e pdQSM-EXTRACTION na decomposição de arquivos de tamanho reduzido, mais precisamente na faixa de 1 a 7MB.

5.2.1 Cenário 1: Decomposição de Arquivos Pequenos

O primeiro cenário teve por finalidade analisar o desempenho das estratégias QSM-EXTRACTION e pdQSM-EXTRACTION na decomposição de arquivos de tamanho reduzido, mais precisamente na faixa de 1 a 7MB. Para isso, foram utilizados cinco tipos de arquivos distintos:

- **Arquivos de Música**: Músicas reais em formato **mp3**;

- **Arquivos de Imagem**: Imagens reais em formato **jpg**;
- **Arquivos de Texto**: Textos sintéticos em formato **txt**;
- **Sinais de Eletrocardiograma (ECG)**: Arquivos contendo sinais de exames de eletrocardiograma, em formato binário **extensão .dat**;
- **Arquivos de Vídeos**: Vídeos reais, em formato **mp4**.

Neste primeiro experimento utilizamos arquivos de diferentes tipos (Música, Imagem, Texto, Sinal e Vídeo) com tamanhos variando de 1 a 7.1MB. Para cada um desses arquivos executamos o processo de decomposição dez vezes e calculamos a média do tempo de entrada (tempo gasto na decomposição). Esse processo foi executado para cada um dos algoritmos avaliados: QSM-EXTRACTION, pdQSM-EXTRACTIONv1, pdQSM-EXTRACTIONv2 e pdQSM-EXTRACTIONv3. Neste cenário, foram utilizados um nó *Master* e apenas dois nós escravos.

A Figura 26 ilustra os tempo de entrada (em segundos), no eixo Y, obtidos pelas quatro estratégias analisadas. No eixo X tem-se os arquivos utilizados, destacando-se o seu tipo e o seu tamanho. Desta forma, "T-0,5" representa um arquivo texto com tamanho de 0,5MB, "I-1,4" indica um arquivo de imagem contendo 1,4MB, "M-1,7" simboliza um arquivo de música (mp3) de tamanho 1,7MB, "S-1,9" descreve um arquivo de sinal (ECG) contendo 1,9MB e "V-3,8" denota um arquivo de vídeo de tamanho 3,8MB. Analisando a Figura 26 pode-se concluir que, para arquivos pequenos, a estratégia QSM-EXTRACTION apresenta um tempo de entrada menor para todos os tipos de arquivo investigados. Assim, para este cenário, a utilização das versões paralelas baseadas no paradigma *MapReduce* não se mostrou vantajosa. Além disso, podemos observar que dentre as versões paralelas, a estratégia pdQSM-EXTRACTIONv3 apresentou o pior desempenho, o que se justifica pela compressão executada sobre os três arquivos resultantes do processo de decomposição.

A Figura 27 ilustra a razão entre a soma dos tamanhos dos arquivos gerados na decomposição (contendo qualidade, quantidade e medida) e o tamanho do arquivo original. Pode-se observar que a estratégia QSM-EXTRACTION apresenta um aumento constante de 40% em relação ao tamanho do arquivo original, independentemente do tipo de arquivo. Este resultado mantém relação com os resultados reportados nos artigos que propuseram a estratégia QSM-EXTRACTION. A estratégia pdQSM-EXTRACTIONv1 apresenta aumentos, em relação ao tamanho do arquivo original, entre 200% e 300%, dependendo do tipo de arquivo. Vale destacar que a estratégia pdQSM-EXTRACTIONv1 utiliza posição global e realiza compressão

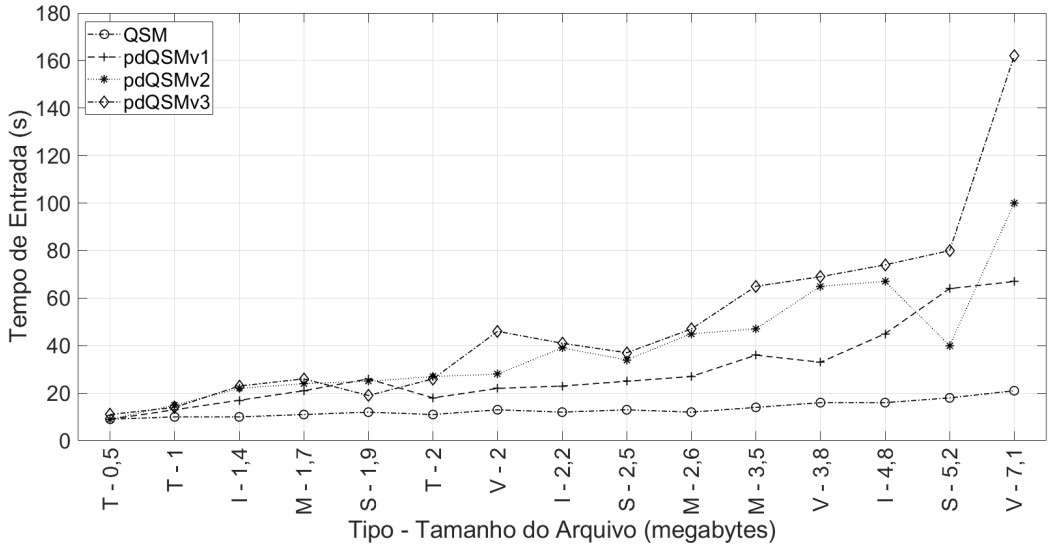

Figura 26 – Tempo de Entrada (Decomposição) para Arquivos Pequenos

apenas sobre o arquivo que contém a medida. Portanto, espera-se que a soma dos arquivos contendo a qualidade, a quantidade e a medida seja sempre maior que o tamanho original do arquivo. Já a estratégia pdQSM-EXTRACTIONv2 utiliza posição local e, assim como pdQSM-EXTRACTIONv1, realiza compressão apenas sobre o arquivo que contém a medida. Logo, espera-se que a estratégia pdQSM-EXTRACTIONv2 apresente resultados melhores que pdQSM-EXTRACTIONv1 na métrica analisada na Figura 27. Contudo, observa-se que para imagens, músicas e vídeos a estratégia pdQSM-EXTRACTIONv2 apresentou resultados piores que pdQSM-EXTRACTIONv1. Este resultado é explicado pela entropia, ou seja, pela quantidade de *bytes* diferentes. A entropia das imagens, músicas e vídeos analisadas era maior que a entropia dos arquivos de texto utilizados. Por exemplo, T-1 é um texto com muita repetição, ou seja, muitos *bytes* repetidos, ou seja, entropia baixa. Analisando as estratégias pdQSM-EXTRACTIONv2 e pdQSM-EXTRACTIONv3 observa-se que ambas possuem um comportamento bastante semelhante, o que é esperado uma vez que ambas utilizam posição local. Contudo, pdQSM-EXTRACTIONv3 mantém sempre valores mais baixos que pdQSM-EXTRACTIONv2 para a métrica analisada na Figura 27. Este resultado decorre do fato de pdQSM-EXTRACTIONv3 aplicar compressão nos arquivos contendo qualidade, quantidade e medida, enquanto pdQSM-EXTRACTIONv2 aplica compressão apenas na medida.

A Figura 28 destaca os tempo de entrada (em segundos) para arquivos do tipo texto. Pode-se observar que, para arquivos texto, a estratégia QSM-EXTRACTION apresenta os melhores resultados. Vale destacar que entre as três estratégias propostas nesta dissertação, a pdQSM-EXTRACTIONv1 apresentou o melhor desempenho.

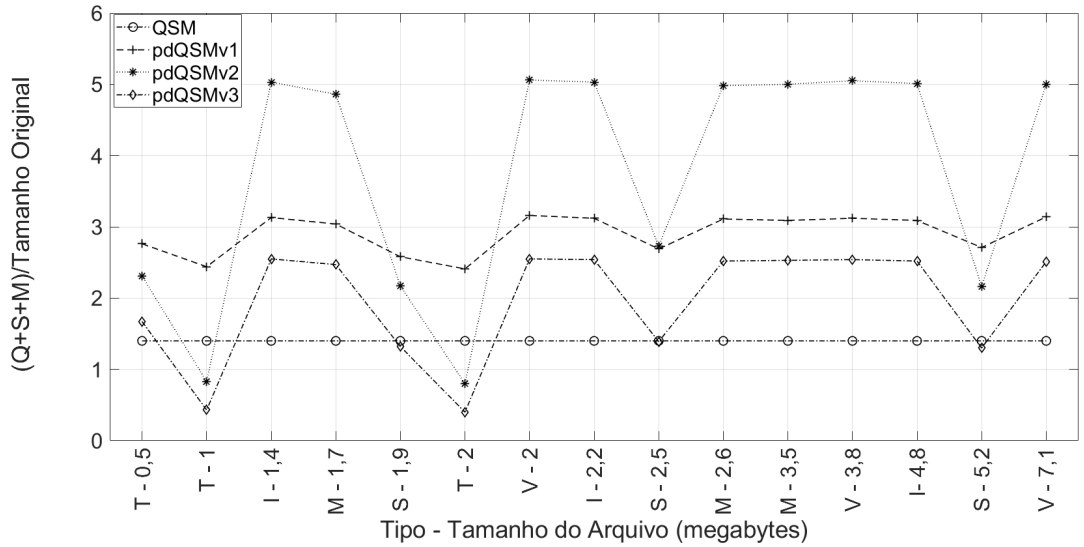

Figura 27 – Razão entre a Soma dos Tamanhos dos Arquivos Gerados na Decomposição (Q + S + M) e Tamanho do Arquivo Original

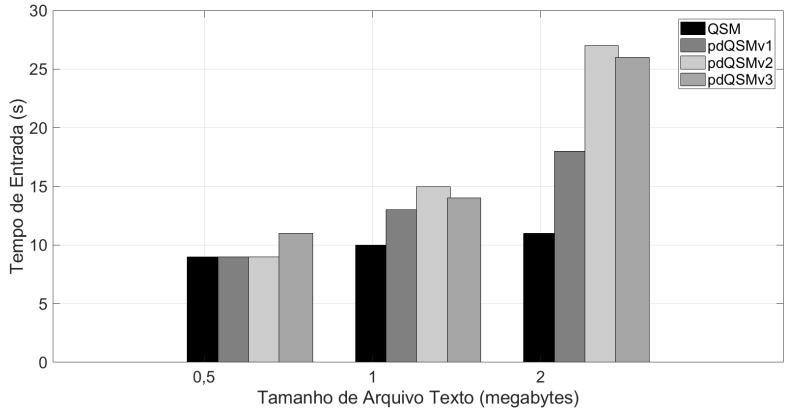

Figura 28 – Tempo de Entrada (Decomposição) para Arquivos do Tipo Texto

A Figura 29 destaca a razão entre a soma dos tamanhos dos arquivos gerados na decomposição (contendo qualidade, quantidade e medida) e o tamanho do arquivo original para arquivos do tipo texto. Pode-se observar por meio da Figura 29 que, à medida que o tamanho do arquivo de texto cresce, a estratégia pdQSM-EXTRACTIONv3 passa a proporcionar os melhores resultados, uma vez esta aplica compressão nos arquivos contendo qualidade, quantidade e medida.

A Figura 30 destaca os tempo de entrada (em segundos) para arquivos do tipo imagem. Pode-se observar que, para arquivos de imagem, a estratégia QSM-EXTRACTION apresenta os melhores resultados. Vale destacar que entre as três estratégias propostas nesta dissertação, a pdQSM-EXTRACTIONv1 apresentou o melhor desempenho.

A Figura 31 destaca a razão entre a soma dos tamanhos dos arquivos gerados na decomposição (contendo qualidade, quantidade e medida) e o tamanho do arquivo original

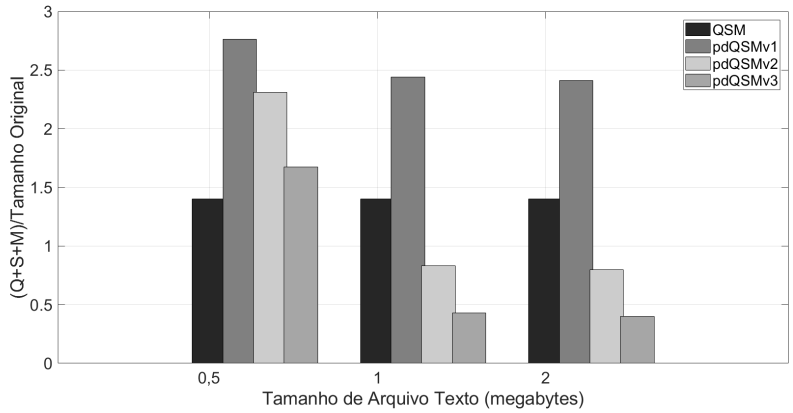

Figura 29 – Razão entre a Soma dos Tamanhos dos Arquivos Gerados na Decomposição (Q + S + M) e Tamanho do Arquivo Original para Textos

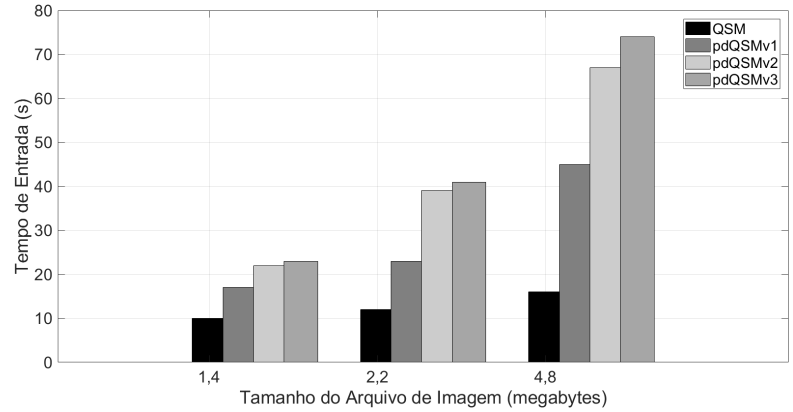

Figura 30 – Tempo de Entrada (Decomposição) para Arquivos do Tipo Imagem

para arquivos do tipo imagem. Pode-se observar por meio da Figura 31 que a estratégia QSM-EXTRACTION apresenta os melhores resultados. Já entre as três estratégias proposta nesta dissertação, pdQSM-EXTRACTIONv1 apresentou o melhor resultado. Além disso, pode-se observar que a estratégia pdQSM-EXTRACTIONv2 apresentou resultados piores que pdQSM-EXTRACTIONv1. Este resultado é explicado pela alta entropia das imagens utilizadas.

A Figura 32 destaca os tempo de entrada (em segundos) para arquivos do tipo música. Pode-se observar que, para arquivos de música, a estratégia QSM-EXTRACTION apresenta os melhores resultados. Vale destacar que entre as três estratégias propostas nesta dissertação, a pdQSM-EXTRACTIONv1 apresentou o melhor desempenho.

A Figura 33 destaca a razão entre a soma dos tamanhos dos arquivos gerados na decomposição (contendo qualidade, quantidade e medida) e o tamanho do arquivo original para arquivos do tipo música. Pode-se observar por meio da Figura 31 que a estratégia QSM-EXTRACTION apresenta os melhores resultados. Já entre as três estratégias proposta nesta dissertação, pdQSM-EXTRACTIONv3 apresentou o melhor resultado. Além disso, pode-se

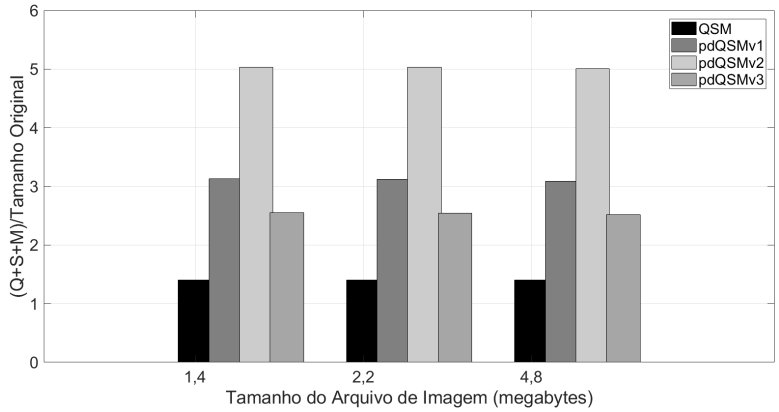

Figura 31 – Razão entre a Soma dos Tamanhos dos Arquivos Gerados na Decomposição (Q + S + M) e Tamanho do Arquivo Original para Imagens

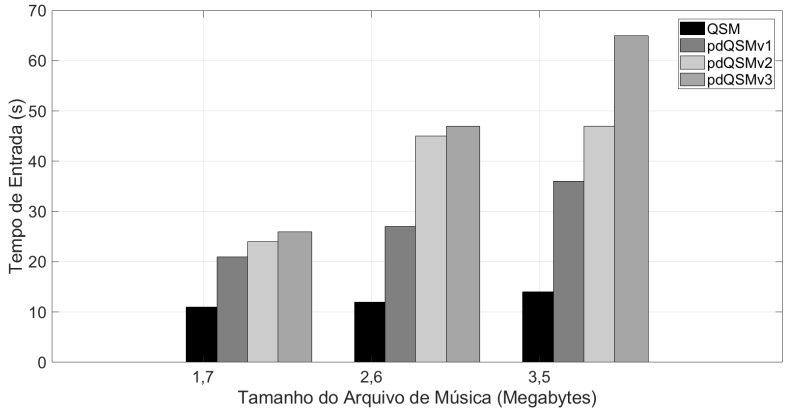

Figura 32 – Tempo de Entrada (Decomposição) para Arquivos do Tipo Música

observar que a estratégia pdQSM-EXTRACTIONv2 apresentou resultados piores que pdQSM-EXTRACTIONv1. Este resultado é explicado pela alta entropia das músicas utilizadas.

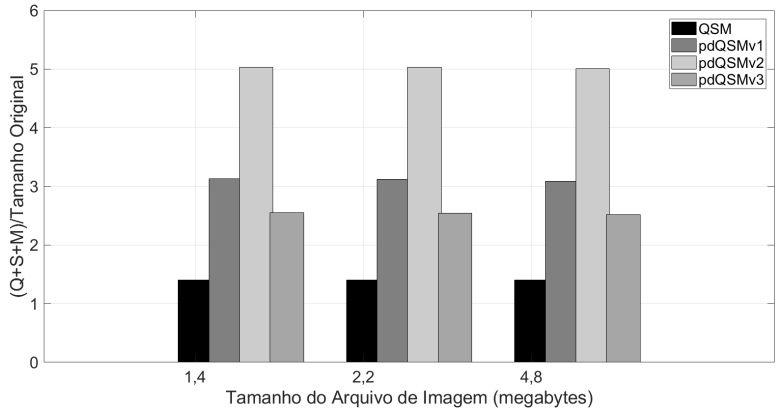

Figura 33 – Razão entre a Soma dos Tamanhos dos Arquivos Gerados na Decomposição (Q + S + M) e Tamanho do Arquivo Original para Músicas

A Figura 34 destaca os tempo de entrada (em segundos) para arquivos contendo sinais ECG. Pode-se observar que, para arquivos de sinais, a estratégia QSM-EXTRACTION

apresenta os melhores resultados. Vale destacar que entre as três estratégias propostas nesta dissertação, a pdQSM-EXTRACTIONv1 apresentou o melhor desempenho.

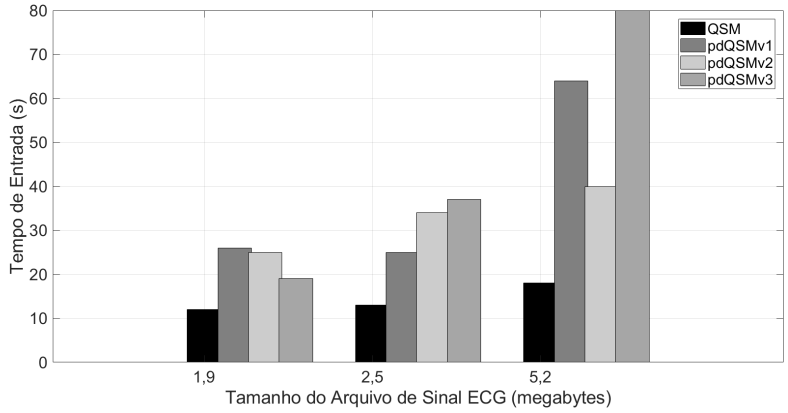

Figura 34 – Tempo de Entrada (Decomposição) para Arquivos de Sinais

A Figura 35 destaca a razão entre a soma dos tamanhos dos arquivos gerados na decomposição (contendo qualidade, quantidade e medida) e o tamanho do arquivo original para arquivos contendo sinais ECG. Pode-se observar por meio da Figura 35 que a estratégia QSM-EXTRACTION apresenta os melhores resultados. Já entre as três estratégias proposta nesta dissertação, pdQSM-EXTRACTIONv3, uma vez esta aplica compressão nos arquivos contendo qualidade, quantidade e medida.

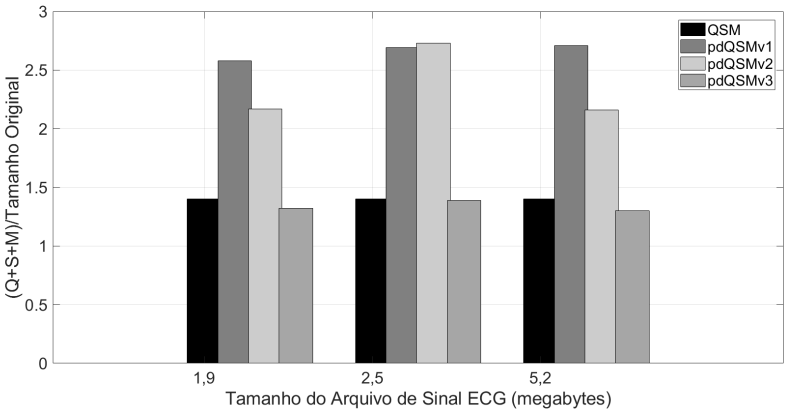

Figura 35 – Razão entre a Soma dos Tamanhos dos Arquivos Gerados na Decomposição (Q + S + M) e Tamanho do Arquivo Original para Sinais

A Figura 36 destaca os tempo de entrada (em segundos) para arquivos de vídeo. Pode-se observar que, para arquivos de vídeo, a estratégia QSM-EXTRACTION apresenta os melhores resultados. Vale destacar que entre as três estratégias propostas nesta dissertação, a pdQSM-EXTRACTIONv1 apresentou o melhor desempenho.

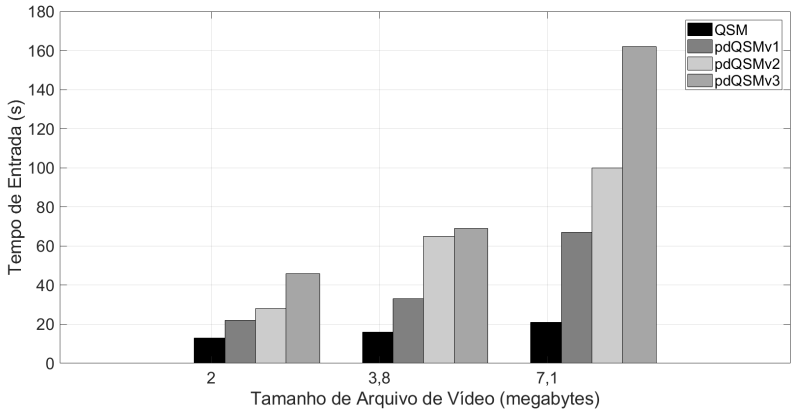

Figura 36 – Tempo de Entrada (Decomposição) para Arquivos de Vídeo

A Figura 37 destaca a razão entre a soma dos tamanhos dos arquivos gerados na decomposição (contendo qualidade, quantidade e medida) e o tamanho do arquivo original para arquivos de vídeo. Pode-se observar por meio da Figura 37 que a estratégia QSM-EXTRACTION apresenta os melhores resultados. Já entre as três estratégias proposta nesta dissertação, pdQSM-EXTRACTIONv3 apresentou o melhor resultado. Além disso, pode-se observar que a estratégia pdQSM-EXTRACTIONv2 apresentou resultados piores que pdQSM-EXTRACTIONv1. Este resultado é explicado pela alta entropia dos vídeos utilizadas.

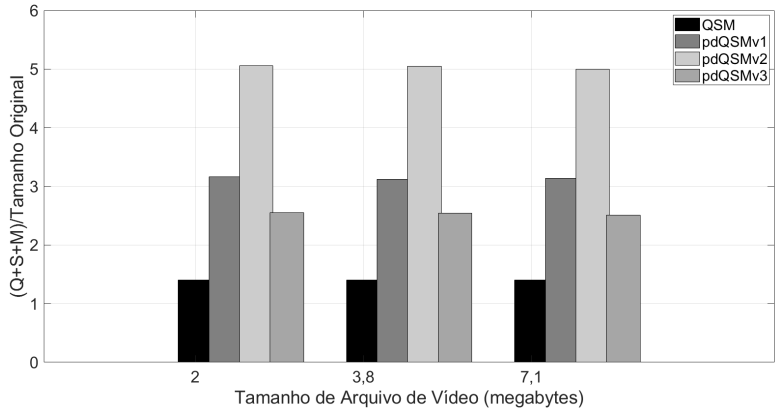

Figura 37 – Razão entre a Soma dos Tamanhos dos Arquivos Gerados na Decomposição (Q + S + M) e Tamanho do Arquivo Original para Vídeos

Como pudemos observar neste experimento, a entropia dos arquivos proporciona um impacto significativo no tamanho dos arquivos gerados no processo de decomposição para armazenar as características de qualidade, quantidade e medida. Este fato decorre da estratégia utilizada para ordenar os blocos, posição global e posição local. Além disso, a entropia também proporciona um impacto significativo no processo de compressão. Arquivos com baixa entropia proporcionam maiores níveis de compactação. Com a finalidade de avaliar este impacto,

calculamos a entropia dos arquivos utilizados neste cenário. Para isso utilizamos a entropia de e Shannon. A Tabela 2 ilustra os resultados obtidos. Como supomos anteriormente, os arquivos de texto possuem uma baixa entropia, os arquivos de sinais uma entropia média, enquanto os arquivos de imagem, música e vídeo possuem os maiores valores para a entropia.

Tabela 2 – Entropia dos Arquivos Utilizados

Arquivo	Entropia
T - 0,5	5,03
T - 1	3,22
T - 2	3,22
S - 1,9	5,47
S - 2,5	6,45
S - 5,2	5,54
I - 1,4	7,97
I - 2,2	7,97
I - 4,8	7,96
M - 1,7	7,91
M - 2,6	7,92
M - 3,5	7,96
V - 2	7,99
V - 3,8	7,99
V - 7,1	7,99

5.2.2 Cenário 2: Decomposição de Arquivos Grandes

O segundo cenário teve por finalidade analisar o desempenho das estratégias QSM-EXTRACTION e pdQSM-EXTRACTION na decomposição de arquivos de tamanho elevado, mais precisamente na faixa de 1 a 10 GB. Para isso, foram utilizados arquivos de texto sintéticos em formato **txt**. Neste segundo experimento utilizamos arquivos de diferentes tamanhos, variando de 1 a 10GB. Para cada um desses arquivos executamos o processo de decomposição dez vezes e calculamos a média do tempo de entrada (tempo gasto na decomposição). Esse processo foi executado para cada um dos algoritmos avaliados: QSM-EXTRACTION e pdQSM-EXTRACTIONv1. Dentre as três estratégias propostas nesta dissertação, apenas a pdQSM-EXTRACTIONv1 foi analisada neste cenário, uma vez que pelo primeiro cenário comprovou-se que esta apresenta o melhor desempenho. Neste segundo cenário, também foram utilizados um nó *Master* e apenas dois nós escravos.

A Figura 38 ilustra os tempo de entrada (em segundos), no eixo Y, obtidos pelas duas estratégias analisadas. Já o eixo X destaca o tamanho dos arquivos utilizados. Analisando

a Figura 38 pode-se concluir que para arquivos com tamanho de até 2GB a estratégia QSM-EXTRACTION é mais eficiente. Já para arquivos com tamanhos maiores ou iguais a 4GB a estratégia pdQSM-EXTRACTIONv1 apresentou melhor desempenho. Neste sentido, podemos identificar um limiar, em relação ao tamanho dos arquivos, a partir do qual a utilização do paradigma *MapReduce* se mostra vantajosa.

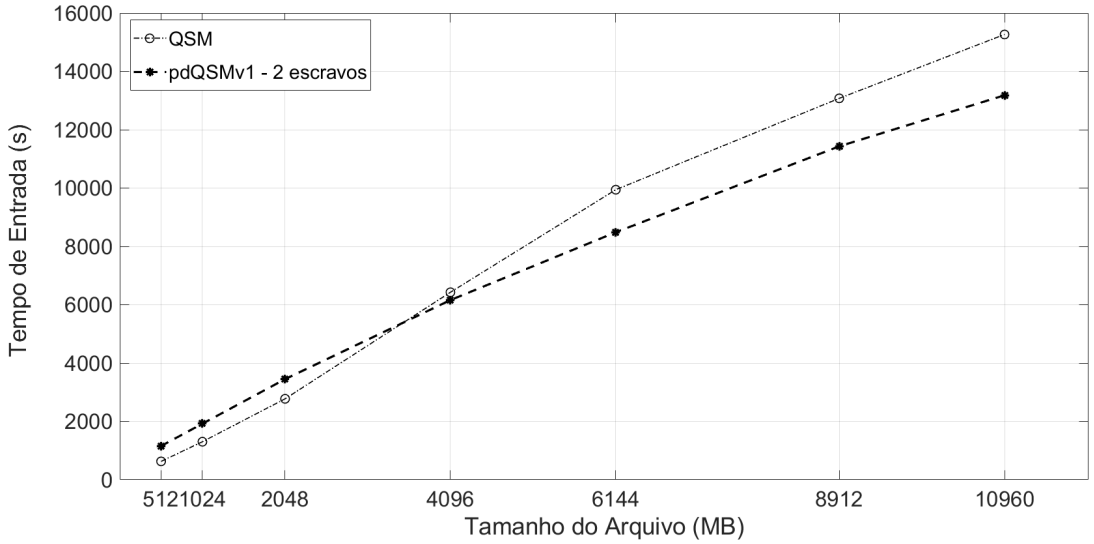

Figura 38 – Tempo de Entrada (Decomposição) para Arquivos Grandes

A Tabela 3 apresenta os valores absolutos obtidos neste segundo experimento. Observe que para arquivos com tamanhos de 4GB o tempo de entrada obtido pela estratégia pdQSM-EXTRACTIONv1 é cerca de 95,83% do tempo de entrada obtido pela estratégia QSM-EXTRACTION, proporcionando um ganho de aproximadamente 4%. Já para arquivos de 6GB o tempo de entrada obtido pela estratégia pdQSM-EXTRACTIONv1 é cerca de 85,33% do tempo de entrada obtido pela estratégia QSM-EXTRACTION, proporcionando um ganho de aproximadamente 15%. Vale destacar também que este ganho se mantém praticamente estável para arquivos com 8 e 10GB. Assim, seria razoável supor que a estratégia pdQSM-EXTRACTIONv1 não apresentaria ganhos maiores que 15% para arquivos maiores de 10GB, utilizando o mesmo ambiente de experimentação, composto por um *Master* e dois nós escravos.

5.2.3 Cenário 3: Impacto da Adição de um Nó Escravo

No terceiro cenário, adicionamos mais um nó escravo na estrutura do *cluster*. Nossa hipótese é que a adição de nós escravos possa fazer com que a estratégia pdQSM-EXTRACTIONv1 apresente ganhos superiores a 15%. Neste cenário, foram utilizados arquivos

Tabela 3 – Tempo de Entrada (segundos): QSM-EXTRACTION *versus* pdQSM-EXTRACTIONv1

Tamanho	512 MB	1024 MB	2048 MB	4096 MB	6144 MB	8912 MB	10960 MB
QSM	630	1304	2775	6427	9943	13075	15265
pdQSM	1157	1922	3450	6159	8484	11430	13180
Razão (%): pdQSM / QSM	183.65%	147.39%	124.32%	95.83%	85.33%	87.42%	86.34%

de texto sintéticos em formato **txt**, com tamanhos variando entre 1 e 10GB. Para cada um desses arquivos executamos o processo de decomposição dez vezes e calculamos a média do tempo de entrada (tempo gasto na decomposição). Esse processo foi executado para cada um dos algoritmos avaliados: QSM-EXTRACTION e pdQSM-EXTRACTIONv1. Contudo, neste último cenário foram utilizados um nó *Master* e três nós escravos. O objetivo deste experimento foi avaliar o impacto da adição de um nó escravo no desempenho da estratégia pdQSM-EXTRACTIONv1.

A Figura 39 sintetiza os resultados obtidos neste terceiro cenário de testes. No eixo Y observa-se os tempo de entrada (em segundos) obtidos pelas duas estratégias analisadas. Já o eixo X destaca o tamanho dos arquivos utilizados. Analisando a Figura 39 pode-se concluir que para arquivos com tamanho de até 2GB a estratégia QSM-EXTRACTION é mais eficiente. Já para arquivos com tamanhos maiores ou iguais a 4GB a estratégia pdQSM-EXTRACTIONv1 apresentou melhor desempenho. Apesar deste resultado ser semelhante ao que foi obtido no cenário anterior, perceba que a diferença entre os tempos de entrada das estratégias pdQSM-EXTRACTIONv1 e QSM-EXTRACTION se acentua bastante. Neste sentido, podemos identificar concluir que a adição de um nó escravo proporcionou um ganho de desempenho considerável à estratégia pdQSM-EXTRACTIONv1.

A Tabela 5 apresenta os valores absolutos obtidos neste terceiro experimento. Observe que para arquivos com tamanhos de 4GB o tempo de entrada obtido pela estratégia pdQSM-EXTRACTIONv1 é cerca de 61,29% do tempo de entrada obtido pela estratégia QSM-EXTRACTION, proporcionando um ganho de aproximadamente 39%. Já para arquivos de 6GB o tempo de entrada obtido pela estratégia pdQSM-EXTRACTIONv1 é cerca de 53,57% do tempo de entrada obtido pela estratégia QSM-EXTRACTION, proporcionando um ganho de aproximadamente 46%. Vale destacar também que este ganho se mantém entre 39% e 46% para arquivos com 8 e 10GB, apresentando um comportamento praticamente estável.

Finalmente, podemos comparar o desempenho da estratégia pdQSM-EXTRACTIONv1

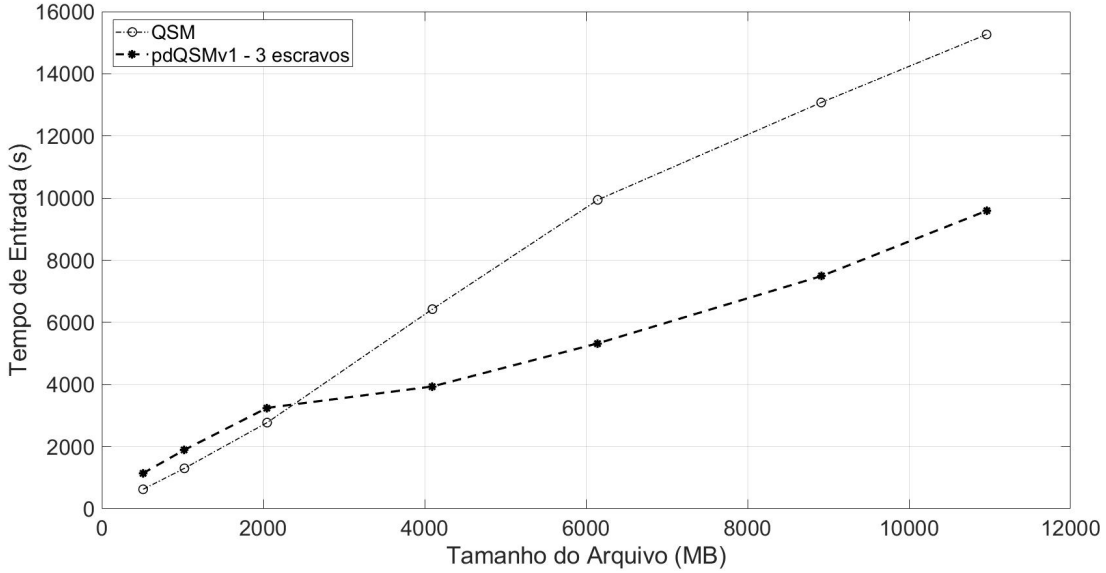

Figura 39 – Tempo de Entrada (segundos): QSM-EXTRACTION *versus* pdQSM-EXTRACTIONv1 (Usando 3 Nós Escravos)

Tabela 4 – Tempo de Entrada (segundos): QSM-EXTRACTION *versus* pdQSM-EXTRACTIONv1 (Usando 3 Nós Escravos)

Tamanho	512 MB	1024 MB	2048 MB	4096 MB	6144 MB	8912 MB	10960 MB
QSM	630	1304	2775	6427	9943	13075	15265
PD-QSM	1148	1899	3250	3939	5326	7489	9590
Razão (%): pdQSM / QSM	182.22%	145.63%	117.12%	61.29%	53.57%	57.28%	62.82 %

usando 2 e 3 nós escravos. A Figura 40 ilustra o resultado desta comparação. Pode-se observar por meio da Figura 40 que para arquivos com tamanho até 2GB o fato de utilizar um nó a mais não proporcionou maiores impactos no desempenho da estratégia pdQSM-EXTRACTIONv1. Porém, para arquivos maiores que 2GB podemos observar um ganho expressivo, na faixa de 30%.

A Tabela 5 apresenta os valores absolutos obtidos pela estratégia pdQSM-EXTRACTIONv1 com 2 e com 3 nós escravos. Observe que para arquivos com tamanhos de 4GB o tempo de entrada obtido pela estratégia pdQSM-EXTRACTIONv1 com 3 escravos é cerca de 63,96% do tempo de entrada obtido pela estratégia pdQSM-EXTRACTIONv1 com 2 escravos, proporcionando um ganho de aproximadamente 36%. Já para arquivos de 10GB, o tempo de entrada obtido pela estratégia pdQSM-EXTRACTIONv1 com 3 escravos é cerca de 72,76% do tempo de entrada obtido pela estratégia pdQSM-EXTRACTIONv1 com 2 nós escravos, proporcionando um ganho de aproximadamente 27%. Vale destacar também que este ganho se mantém entre 27% e 36% para arquivos com 6 e 8GB, apresentando um comportamento praticamente estável.

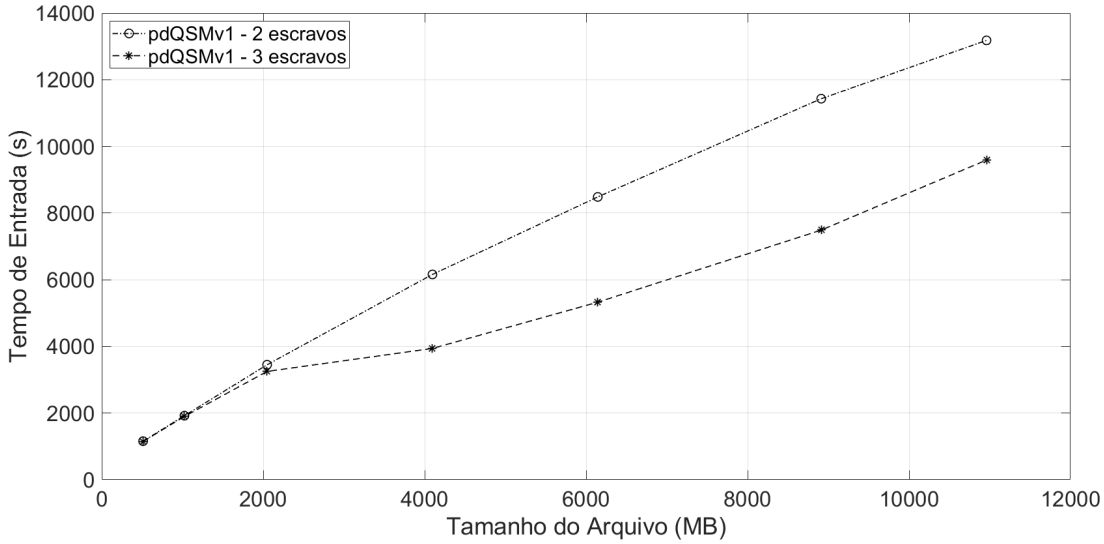

Figura 40 – Tempo de Entrada: pdQSM-EXTRACTION com 2 Escravos *versus* pdQSM-EXTRACTION com 3 Escravos

Tabela 5 – Tempo de Entrada: pdQSM-EXTRACTION com 2 Escravos *versus* pdQSM-EXTRACTION com 3 Escravos

Tamanho	512 MB	1024 MB	2048 MB	4096 MB	6144 MB	8912 MB	10960 MB
pdQSM	1157	1922	3450	6159	8484	11430	13180
pdQSM +1 slave	1148	1899	3250	3939	5326	7489	9590
Razão (%): (pdQSM +1 slave) / pdQSM	99.22%	98.80%	94.20%	63.96%	62.78%	65.52%	72.76 %

5.3 Conclusão

Neste capítulo, apresentamos os experimentos realizados e os resultados obtidos, com a finalidade de demonstrar a eficiência das ideias que norteiam a estratégia proposta nesta dissertação. Os resultados experimentais nos dão suporte para a comparação das nossas contribuições face aos trabalhos existentes na literatura relacionada.

6 CONCLUSÃO E TRABALHOS FUTUROS

6.1 Conclusão

Atualmente, o armazenamento de grandes quantidades de dados confidenciais em servidores na nuvem é uma prática comum, uma vez que permite reduzir custos e aumentar a disponibilidade desses dados. Porém, nos ambientes de computação em nuvem, o controle dos dados deixa de ser do seu proprietário e passa a ser do provedor do serviço de armazenamento, o que faz surgir novos desafios relacionados à privacidade, segurança e confidencialidade.

Neste contexto, diferentes soluções para assegurar a confidencialidade dos dados armazenados na nuvem foram propostas. Em geral, tais estratégias utilizam criptografia, fragmentação de dados ou uma combinação dessas duas abordagens. Recentemente, uma nova solução, denominada QSM-EXTRACTION, foi proposta. A estratégia QSM-EXTRACTION baseia-se na fragmentação de um arquivo digital em fragmentos denominados objetos de informação, na decomposição desses objetos por meio da extração de suas características e na dispersão dessas características em diferentes serviços de armazenamento em nuvem.

Contudo, apesar de ter sido concebida para o ambiente de computação em nuvem, a QSM-EXTRACTION adota uma abordagem de execução centralizada, o que pode comprometer o desempenho da etapa de decomposição. Neste sentido, propomos uma extensão, paralela e distribuída, da estratégia QSM-EXTRACTION, denominada pdQSM-EXTRACTION, a qual explora o paradigma MapReduce com a finalidade de possibilitar uma maior eficiência no processo de extração das características dos objetos de informação. Adicionalmente, foram implementadas e avaliadas três versões diferentes da estratégia pdQSM-EXTRACTION, denominadas: pdQSM-EXTRACTIONv1, pdQSM-EXTRACTIONv2 e pdQSM-EXTRACTIONv3. Essas três versões da estratégia pdQSM-EXTRACTION foram implementadas em linguagem Scala utilizando-se o *framework Apache Spark*.

Diversos experimentos foram realizados para avaliar a estratégia proposta. Para executar estes experimentos, foi utilizada uma infraestrutura de nuvem pública gerida pelo Amazon. Três diferentes cenários de execução foram avaliados. Os resultados obtidos comprovam a viabilidade da utilização da estratégia pdQSM-EXTRACTION em aplicações envolvendo grandes volumes de dados. A nossa hipótese inicial era que a partir de um determinado tamanho de arquivo a utilização do paradigma *MapReduce*, por meio da execução paralela e da distribuição dos dados, proporcionaria melhores desempenhos na fase de decomposição da estratégia QSM-

EXTRACTION, em comparação com sua execução centralizada. Os resultados dos experimentos realizados comprovaram a nossa hipótese. A análise dos resultados também permite concluir que a versão pdQSM-EXTRACTIONv1, a qual utiliza posição global, apresentou, na maioria dos cenários, o menor tempo de execução. Por outro lado, as versões pdQSM-EXTRACTIONv2 e pdQSM-EXTRACTIONv3, as qausi utilizam posição local, geraram arquivos mais compactos para armazenar as características de qualidade, quantidade e medida.

Por fim, as principais contribuições desta dissertação foram:

1. Uma nova estratégia paralela e distribuída, denominada pdQSM-EXTRACTION, para assegurar a confidencialidade de dados em serviços de armazenamento em nuvem;
2. Utilização do paradigma MapReduce;
3. Implementação da estratégia pdQSM-EXTRACTION;
4. Avaliação da estratégia pdQSM-EXTRACTION;

6.2 Trabalhos Futuros

Existem oportunidades de trabalhos futuros a partir da pesquisa realizada nesta dissertação. Inicialmente, podemos avaliar a utilização de outros tipos de compressão de dados, com a possibilidade de melhorar o desempenho e diminuir o tamanho dos arquivos gerados. Adicionalmente, podemos investigar a utilização de outras estruturas de dados do *spark*, como *dataframes*, *datasets* e *streaming*, sem comprometer o funcionamento e a corretude da estratégia proposta. Por fim, a inserção de descentralização no processo de recomposição (estratégia distribuída), a possibilidade de recompor apenas uma parte do arquivo e a investigação de outros paradigmas de processamento paralelo, como, por exemplo, o uso de GPUs (*Graphics Processing Units* ou Unidades de Processamento Gráfico).

REFERÊNCIAS

AGGARWAL, C. C. On k-anonymity and the curse of dimensionality. In: **Proceedings of the 31st international conference on Very large data bases**. [S.l.]: VLDB Endowment, 2005. p. 901–909.

BARDIN, J.; CALLAS, J.; CHAPUT, S.; FUSCO, P.; GILBERT, F.; HOFF, C.; HURST, D.; KUMARASWAMY, S.; LYNCH, L.; MATSUMOTO, S. *et al.* Security guidance for critical areas of focus in cloud computing. **Cloud Security Alliance**, p. 0–176, 2009.

BRANCO, E. C.; MACHADO, J.; MARIA, J. A strategy to preserve data confidentiality in cloud storage services. **Workshop de Teses e Dissertações em Banco de Dados (WTDBD)**, 2014.

BRANCO, E. C.; MACHADO, J.; MARIA, J. Uma estratégia para assegurar a confidencialidade de dados armazenados em nuvem. **Universidade Federal do Ceará (UFC)**, 2017.

BRANCO, E. C.; MARIA, J.; MACHADO, J.; REIS, R. A flexible mechanism for data confidentiality in cloud database scenarios. **18th International Conference on Enterprise Information Systems**, 2016.

BRANCO, E. C.; MARIA, J.; MACHADO, J.; REIS, R. A new approach to preserving data confidentiality in the clouds. **Proceedings of the 20th International Database Engineering & Applications Symposium**, 2016.

BRANCO, E. C.; MONTEIRO, J. M.; REIS, R.; MACHADO, J. C. A new mechanism to preserving data confidentiality in cloud database scenarios. In: SPRINGER. **International Conference on Enterprise Information Systems**. [S.l.], 2016. p. 261–283.

BUYYA, R.; BROBERG, J.; GOSCINSKI, A. M. **Cloud computing: Principles and paradigms**. [S.l.]: John Wiley & Sons, 2010. v. 87.

CESELLI, A.; DAMIANI, E.; VIMERCATI, S. D. C. D.; JAJODIA, S.; PARABOSCHI, S.; SAMARATI, P. Modeling and assessing inference exposure in encrypted databases. **ACM Transactions on Information and System Security (TISSEC)**, v. 8, n. 1, p. 119–152, 2005.

CHEN, D.; ZHAO, H. Data security and privacy protection issues in cloud computing. In: IEEE. **Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on**. [S.l.], 2012. v. 1, p. 647–651.

CIRIANI, V.; VIMERCATI, S. D. C. D.; FORESTI, S.; JAJODIA, S.; PARABOSCHI, S.; SAMARATI, P. Keep a few: Outsourcing data while maintaining confidentiality. In: **Proceedings of the 14th European Conference on Research in Computer Security**. Springer-Verlag, 2009. (ESORICS'09), p. 440–455. ISBN 3-642-04443-3, 978-3-642-04443-4. Disponível em: <http://dl.acm.org/citation.cfm?id=1813084.1813120>.

CIRIANI, V.; VIMERCATI, S. D. C. D.; FORESTI, S.; JAJODIA, S.; PARABOSCHI, S.; SAMARATI, P. Combining fragmentation and encryption to protect privacy in data storage. **ACM Transactions on Information and System Security (TISSEC)**, ACM, v. 13, n. 3, p. 22, 2010.

CONSORTIUM, U. **The Unicode Standard, Version 3.0**. Addison-Wesley, 2000. (Programming languages, v. 1). ISBN 9780201616330. Disponível em: <https://books.google.com.br/books?id=-u5QAAAAMAAJ>.

DEAN, J.; GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. **Symposium on Operating Systems Design and Implementation(OSDI)**, 2004.

FUGKEAW, S. Achieving privacy and security in multi-owner data outsourcing. In: **Seventh International Conference on Digital Information Management, ICDIM 2012, Macau, Macao, August 22-24, 2012**. [S.l.: s.n.], 2012. p. 239–244.

GROBAUER, B.; WALLOSCHEK, T.; STOCKER, E. Understanding cloud computing vulnerabilities. **IEEE Security & Privacy**, IEEE, v. 9, n. 2, p. 50–57, 2011.

GUTTMAN, B.; ROBACK, E. A. **An introduction to computer security: the NIST handbook**. [S.l.]: DIANE Publishing, 1995.

HON, W. K.; MILLARD, C.; WALDEN, I. The problem of 'personal data'in cloud computing: what information is regulated?—the cloud of unknowing. **International Data Privacy Law**, v. 1, n. 4, p. 211–228, 2011.

HU, T. **A Prehistory of the Cloud**. MIT Press, 2015. 209 p. ISBN 9780262029513. Disponível em: <https://books.google.com.br/books?id=34hkCgAAQBAJ>.

JOSEPH, N. M.; DANIEL, E.; VASANTHI, N. A. Article: Survey on privacy-preserving methods for storage in cloud computing. **IJCA Proceedings on Amrita International Conference of Women in Computing - 2013**, AICWIC, n. 4, p. 1–4, jan. 2013. Full text available.

JUNG, T.; LI, X.-y.; WAN, Z.; WAN, M. Privacy preserving cloud data access with multi-authorities. In: **INFOCOM, 2013 Proceedings IEEE**. [S.l.: s.n.], 2013. p. 2625–2633.

KANTARCIOĞLU, M.; CLIFTON, C. Security issues in querying encrypted data. In: ____. **Data and Applications Security XIX: 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Storrs, CT, USA, August 7-10, 2005. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 325–337. ISBN 978-3-540-31937-5. Disponível em: <http://dx.doi.org/10.1007/11535706_24>.

KARAU, H.; WARREN, R. **High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark**. [S.l.]: O'Reilly Media, 2017.

KEEN, A. **Digital Vertigo: How Today's Online Social Revolution Is Dividing, Diminishing, and Disorienting Us**. St. Martin's Press, 2012. ISBN 9780312624989. Disponível em: <https://books.google.com.br/books?id=9T-RgMe9a9wC>.

KRUTZ, R. L.; VINES, R. D. **Cloud security: A comprehensive guide to secure cloud computing**. [S.l.]: Wiley, 2010.

LI, M.; YU, S.; REN, K.; LOU, W.; HOU, Y. T. Toward privacy-assured and searchable cloud data storage services. **IEEE Network**, v. 27, n. 4, p. 56–62, July 2013. ISSN 0890-8044.

LIEBERMAN. **Unisys Security Insights 2015: Consumer viewpoints on security**. [S.l.], 2015.

MELL, P. M.; GRANCE, T. **SP 800-145. The NIST Definition of Cloud Computing**. Gaithersburg, MD, United States, 2011.

NIMGAONKAR, S.; KOTIKELA, S.; GOMATHISANKARAN, M. Ctrust: A framework for secure and trustworthy application execution in cloud computing. In: **Cyber Security (CyberSecurity), 2012 International Conference on**. [S.l.: s.n.], 2012. p. 24–31.

NING, C.; CONG, W.; MING, L.; KUI, R.; WENJING, L. Privacy-preserving multi-keyword ranked search over encrypted cloud data. **Parallel and Distributed Systems, IEEE Transactions on**, v. 25, n. 1, p. 222–233, 2014.

POPA, R. A.; REDFIELD, C. M. S.; ZELDOVICH, N.; BALAKRISHNAN, H. Cryptdb: Protecting confidentiality with encrypted query processing. In: **Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles**. New York, NY, USA: ACM, 2011. (SOSP '11), p. 85–100. ISBN 978-1-4503-0977-6. Disponível em: <http://doi.acm.org/10.1145/2043556.2043566>.

RAGHAVAN, B.; VISHWANATH, K.; RAMABHADRAN, S.; YOCUM, K.; SNOEREN, A. C. Cloud control with distributed rate limiting. **ACM SIGCOMM Computer Communication Review**, ACM, v. 37, n. 4, p. 337–348, 2007.

REKATSINAS, T.; DESHPANDE, A.; MACHANAVAJJHALA, A. Sparsi: Partitioning sensitive data amongst multiple adversaries. **Proc. VLDB Endow.**, VLDB Endowment, v. 6, n. 13, p. 1594–1605, ago. 2013. ISSN 2150-8097. Disponível em: <http://dx.doi.org/10.14778/2536258.2536270>.

RYAN, M. D. Cloud computing privacy concerns on our doorstep. **Commun. ACM**, ACM, New York, NY, USA, v. 54, n. 1, p. 36–38, jan. 2011. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1866739.1866751>.

SAMARATI, P. Data security and privacy in the cloud. In: **Information Security Practice and Experience - 10th International Conference, ISPEC 2014, Fuzhou, China, May 5-8, 2014. Proceedings**. [s.n.], 2014. p. 28–41. Disponível em: <http://dx.doi.org/10.1007/978-3-319-06320-1_4>.

SAMARATI, P.; VIMERCATI, S. D. C. di. Data protection in outsourcing scenarios: Issues and directions. In: **Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security**. ACM, 2010. (ASIACCS '10), p. 1–14. ISBN 978-1-60558-936-7. Disponível em: <http://doi.acm.org/10.1145/1755688.1755690>.

SAYI, T. J. V. R. K.; KRISHNA, R. K. N. S.; MUKKAMALA, R.; BARUAH, P. K. Data outsourcing in cloud environments: A privacy preserving approach. In: **Information Technology: New Generations (ITNG), 2012 Ninth International Conference on**. [S.l.: s.n.], 2012. p. 361–366.

SIEWIOREK, D.; SWARZ, R. **Reliable computer systems: design and evaluation**. Digital Press, 1992. (Computer Technology Series). Disponível em: <https://books.google.com.br/books?id=94lQAAAAMAAJ>.

SOUSA, F.; MOREIRA, L.; MACEDO, J.; MACHADO, J. Gerenciamento de dados em nuvem: Conceitos, sistemas e desafios. **SWIB**, 2010.

SOUSA, F.; MOREIRA, L.; MACHADO, J. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. **ERCEMAP**, 2009.

STALLINGS, W. **Network Security Essentials: Applications and Standards**. 4th. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010. ISBN 0136108059, 9780136108054.

STEFANOV E. SHI, E. Oblivistore: High performance oblivious cloud storage. In: **Security and Privacy (SP), 2013 IEEE Symposium on**. [S.l.: s.n.], 2013. p. 253–267.

TIAN, M.; ZHANG, Y. Analysis of cloud computing and its security. In: **International Symposium on Information Technology in Medicine and Education (ITME)**. [S.l.: s.n.], 2012. (TIME '12).

VENKATARAMANAN, N.; SHRIRAM, A. **Data Privacy: Principles and Practice**. CRC Press, 2016. ISBN 9781498721059. Disponível em: <https://books.google.com.br/books?id=iCANDgAAQBAJ>.

WAMPLER, D.; PAYNE, A. **Programming Scala: Scalability = Functional Programming + Objects**. [S.l.]: OReilly Media, 2014.

WIESE, L. Horizontal fragmentation for data outsourcing with formula-based confidentiality constraints. In: ____. **Advances in Information and Computer Security**. [S.l.]: Springer, 2010. p. 101–116.

XU, X.; XIONG, L.; LIU, J. Database fragmentation with confidentiality constraints: A graph search approach. In: **Proceedings of the 5th ACM Conference on Data and Application Security and Privacy**. ACM, 2015. (CODASPY '15), p. 263–270. ISBN 978-1-4503-3191-3. Disponível em: <http://doi.acm.org/10.1145/2699026.2699121>.

YANG, K.; JIA, X.; REN, K.; ZHANG, B.; XIE, R. Dac-macs: Effective data access control for multiauthority cloud storage systems. **IEEE Transactions on Information Forensics and Security**, v. 8, n. 11, p. 1790–1801, 2013.

YEH, C.-H. A secure shared group model of cloud storage. In: **Proceedings of the 2013 27th International Conference on Advanced Information Networking and Applications Workshops**. Washington, DC, USA: IEEE Computer Society, 2013. (WAINA '13), p. 663–667. ISBN 978-0-7695-4952-1. Disponível em: <http://dx.doi.org/10.1109/WAINA.2013.231>.