6
Implementação

De modo a validar o processo de recomendação de fontes RDF proposto neste trabalho, propomos uma ferramenta que integra diferentes tecnologias da Web semântica. Esta ferramenta implementa uma arquitetura de referência para as ideias aqui expostas.

Neste capítulo é apresentada uma descrição completa da arquitetura que materializa o processo descrito no capítulo 4. Na seção 6.1 são descritos detalhadamente cada um dos componentes da arquitetura. Na seção 6.2 apresenta-se os modelos MapReduce que implementam os fluxos de tarefas para cada uma das etapas do processo de recomendação. Na última seção do capítulo são descritas as opções de otimização das tarefas dos modelos MapReduce.

6.1
Arquitetura

A implementação do sistema foi realizada usando a linguagem de programação Java. A arquitetura é composta por três tipos de componentes, descritos a seguir:

1. Cliente: Reúne basicamente as interfaces necessárias para o usuário interagir, enviando as diferentes requisições de informação tanto aos serviços locais, como aos serviços hospedados na nuvem. Além disso, são necessárias interfaces de acesso os serviços hospedados na Web, como são os diferentes serviços de SPARQL *endPoint* pertencentes as fontes RDF que se poderiam acessar. Também estão incluídos os artefatos necessários para importar um arquivo *dump* com conteúdo RDF encontrados na Web.

2. Web: É composto pelos diferentes serviços de dados disponíveis na Web que fornecem interfaces HTTP de acesso aos dados. Estas interfaces recebem requisições e dão como resposta um conjunto de dados em formato RDF. São consideras nestas interfaces restrições à quantidade de resultados, tempo de espera de uma nova requisição, e o número de

tentativas por requisição. Dentro deste tipo de componente é agrupado também o acesso ao índice semântico na Web, que fornece diferentes formas de acessar os dados.

3. Nuvem: A existência de fornecedores de tecnologia na nuvem [39, 62, 63, 64], permite abstrair conceitos que um usuário comum não precisa conhecer, como por exemplo: A manutenção de um *cluster*, configuração de um servidor Web, ou de banco de dados, e até do mesmo *hardware* que contém especificações complexas necessárias para rodar um aplicativo.

   A nossa arquitetura beneficia-se principalmente do serviço de processamento paralelo e distribuído, que habilita um conjunto de computadores usando o modelo MapReduce sobre o Hadoop *framework*[1] e assim, possibilita o processamento de grandes quantidades de dados. Destacamos que este modelo é independente do fornecedor de tecnologia na nuvem.

   A arquitetura disponibiliza interfaces de comunicação com o serviço de armazenamento na nuvem, direcionados para carregar dados do usuário e manter a informação estatística dos processos MapReduce. Ela também habilita a criação de instancias virtuais para servidores de banco de dados em triplas e de indexação, para ajudar no processo de recomendação de fontes.

Na Figura 6.1 mostramos os componentes da arquitetura e como eles interagem para extrair as diferentes entidades de informação no processo de recomendação de fontes RDF.

6.1.1 Componentes do Cliente

Aplicação Web

Para o desenvolvimento da aplicação Web é usado o padrão Modelo-Vista-Controlador (do inglês *Model-View-Controller* - MVC)[65], que é um dos padrões de arquitetura de software mais utilizados em projetos Web. O Spring Web MVC *framework*[2] é adotado na implementação da aplicação Web, que habilita uma programação consistente em todas os níveis e permite a integração das diferentes camadas da aplicação [66]. O Spring Web MVC *framework* adota no nível da persistência de dados a Hibernate [67][3], que também é usado no desenvolvimento da aplicação Web.

[1] http://hadoop.apache.org/
[2] http://www.springsource.org/
[3] http://www.hibernate.org/

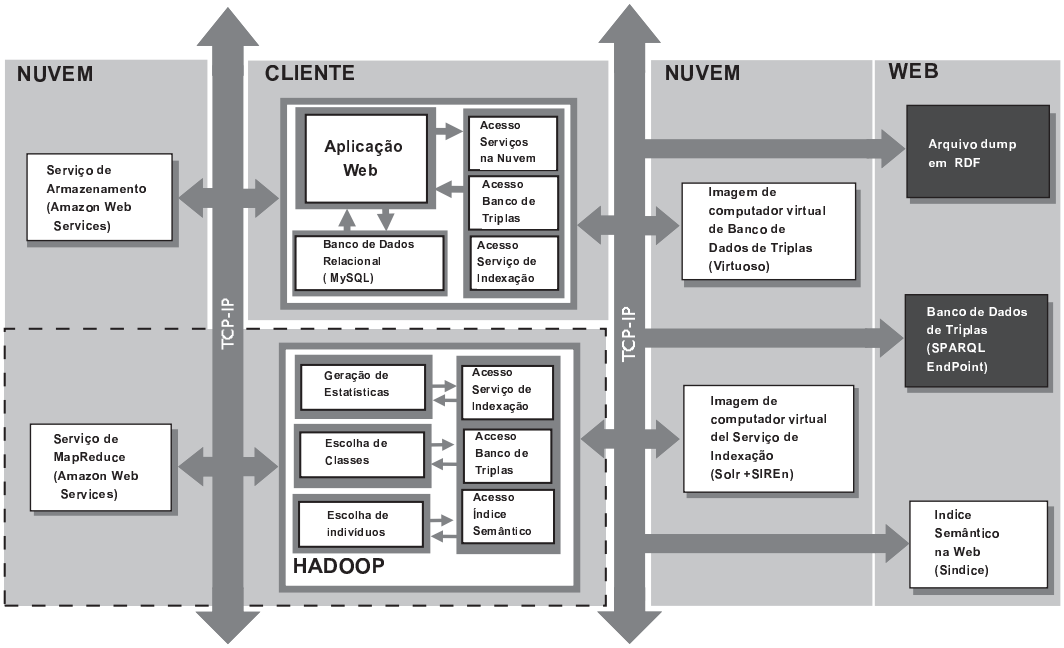

Figura 6.1: Arquitetura proposta

A aplicação Web está composta por diferentes funcionalidades que são agrupadas em módulos, e já foram explicados no capítulo 4, elas descrevem a proposta do processo de recomendação de fontes RDF. Os módulos que compõem à aplicação Web são apresentados na Figura 6.2.

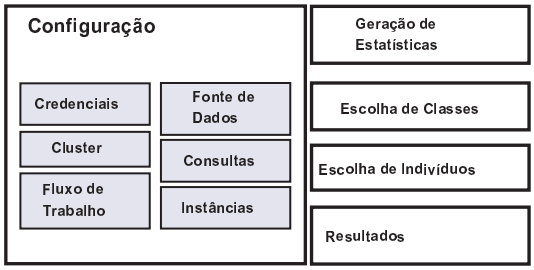

Figura 6.2: Aplicação Web

A seguir descrevemos o módulo da configuração da aplicação Web, que representa o núcleo das funcionalidades oferecidas no processo de recomendação.

1. **Credenciais**: Captura informação da identificação do usuário, esta identificação que habilita o uso dos serviços da nuvem.

2. **Cluster**: Define a descrição do grupo de computadores usados no processamento de dados. Neste módulo, aspectos como a quantidade de computadores, as características do *hardware*, e do *software* usado, são definidos. Pode-se manter diferentes configurações ativas, sem precisar

redefinir nenhuma delas em cada processamento que o usuário precise fazer.

3. **Instâncias**: Este módulo define dois tipos de instâncias (imagens). Um tipo especializado no modo de acesso RDF *dump* e outro de indexação e busca de dados. Este módulo ajuda a definir as características do *hardware* para melhorar o rendimento do processamento das instâncias.

4. **Gerenciamento de Fluxo de tarefa**: Como foi definido na seção 4.1, no capítulo 4, um fluxo de tarefas é definido como: uma sequência de execução de processos MapReduce, um *cluster* e a descrição de diretórios de armazenamento, como foi ilustrado na Figura 4.4. Na seguinte seção são detalhados os diferentes fluxos de execução de tarefas MapReduce usadas pela arquitetura.

5. **Fonte de dados**: Ajuda definir o tipo de acesso à fonte de dados que o usuário escolhe. Para o acesso por RDF *dump*, o aplicativo Web permite ao usuário inserir a URL do arquivo *dump* que deseja analisar e automaticamente ele será carregado no banco de dados de triplas RDF. Este banco de dados é suportado pela instância especializada em dados RDF. Este arquivo *dump* será disponibilizado em um serviço SPARQL *endpoint*.

   Por exemplo, a URL de `http://queens.db.toronto.edu/~oktie/linkedmdb/linkedmdb-latest-dump.zip`, em formato ZIP, oferece informação de filmes do banco de dados LinkedMDB. Este arquivo é descompactado e carregado. Se ele contiver algum formato não suportado pelo banco de dados então eles são desconsiderados no processo. URLs que apontam para arquivos com extensão rdf, xml, nt, owl ou ttl são carregados diretamente no banco de dados de triplas.

   Para o acesso de SPARQL *endpoint*, o usuário precisa apenas indicar a URI do serviço, sem precisar configurar nenhuma instância como *endpoint*.

6. **Consultas**: Permite a gestão de diferentes tipos de consultas usadas durante o processo de recomendação de fontes, parte delas são listadas na tabela 4.1 do capítulo 4. As consultas são organizadas segundo as etapas que elas participam e podem ser modificadas e especializadas pelo usuário.

Interfaces de acesso

O uso de *frameworks* e APIs permite aos desenvolvedores construir aplicações completas com menos esforço de implementação através do reuso de funcionalidades. A seguir, descrevemos as diferentes interfaces de programação que habilitam o acesso aos diferentes recursos usados na arquitetura.

1. **Acesso ao Banco de dados Relacional**: Na seção de Aplicação web 6.1.1, define-se o uso de Hibernate [67] na persistência dos dados. O Hibernate é um *frameworks* ORM (*Object-Relational Mapping*) ou Mapeamento Objeto-Relacional para ambientes Java [68].

   As bibliotecas de Mapeamento Objeto/Relacional fazem o mapeamento de tabelas para classes. Se o banco de dados possui uma tabela chamada **Consulta**, a aplicação provavelmente possuirá uma classe denominada **Consulta**. Essa classe definirá atributos que são usados para receber e alterar os dados dos campos das tabelas, além de métodos para operar sobre os atributos da classe [68].

   O banco de dados relacional usado é MYSQL [4], banco de dados popular em projetos pequenos e na maioria de tipo Web.

2. **Acesso aos Serviço na Nuvem**: Para configurar e usar recursos na nuvem é necessário criar uma interface que abstrai a estrutura dos diferentes serviços que disponibilizam estes recursos.

   No caso do fornecedor de recursos na nuvem *Amazon Web Services*(AWS)[5], ele disponibiliza uma biblioteca que dá acesso aos recursos em diferentes linguagens de programação [39]. Esta biblioteca é adaptada para usar os diferentes serviços que este fornecedor oferece, como por exemplo:

   Amazon Elastic Compute Cloud (EC2): É um serviço da Web que fornece uma capacidade de computação redimensionável na nuvem. Ele define o ambiente virtual com o sistema operacional, os serviços, e a pilha da plataforma de aplicativos exigida para o aplicativo hospedado [69].

   Amazon Simple Storage Service (S3): Fornece uma interface simples de serviço web que pode ser usada para armazenar e recuperar qualquer quantidade de dados, a qualquer momento e de qualquer lugar na Web [69].

[4]http://www.mysql.com/
[5]http://aws.amazon.com/

Amazon Elastic MapReduce(EMR): É um serviço Web que permite às empresas, pesquisadores, analistas de dados e desenvolvedores processar, de modo fácil e econômico, grandes quantidades de dados. Ele utiliza uma estrutura Hadoop [36] hospedada, sendo executada na infraestrutura de escala da Web do Amazon EC2 e Amazon S3 [70].

Esta interface é usada somente com AWS, mas oferece todas as capacidades de se adaptar com qualquer outro fornecedor de serviços na nuvem.

3. **Acesso ao banco de dados em triplas**: Existem diferentes *frameworks* e bibliotecas que dão suporte ao desenvolvimento de aplicações para a Web Semântica como e Mulgara[6], Sesame[7] e Jena[8] [71].

   Para o acesso às fontes de dados RDF, a arquitetura usa Jena [72]. Jena é um *frameworks* para Java e foi desenvolvido nos laboratórios da *Hewlett-Packard*[9] até 2009, desde então, Jena é desenvolvida e suportada por uma comunidade de código aberto (do inglês *open source*).

   Jena representa grafos RDF como um modelo abstrato. Tem suporte para armazenamento do grafo em memória, em banco de dados relacional e em bancos especializados em triplas. Jena suporta RDF, RDFS e OWL e inclui a possibilidade de ler e escrever em formatos RDF/XML, N3 e NTriples [71]. Jena além de dar acesso via APIs aos grafos RDF e OWL, consegue fazer consultas SPARQL distribuídas usando o motor de consultas ARQ [71].

   As capacidades de acesso distribuído, leitura de formatos RDF e suporte de armazenamento em memória são usados pela nossa arquitetura.

4. **Acesso ao serviço de indexação e busca de dados**: O serviço de indexação está suportado por Sorl[10], [47]. O Solr é um mecanismo de indexação e busca textual de código aberto, mantido pela *Apache Software Foudation*[11][35] e baseado na biblioteca Lucene [12]. Ela oferece recursos sofisticados de indexação e busca textual como: busca com operadores booleanos, busca específica por campo, *highlighting* sobre o resultado da busca, paginação do resultado da busca, *facets* sobre o resultado da busca, *caching* de busca, integração com banco de dados

[6] http://www.mulgara.org/
[7] http://www.openrdf.org/
[8] http://jena.apache.org/
[9] http://www.hpl.hp.com/
[10] http://lucene.apache.org/solr/
[11] http://www.apache.org/
[12] http://lucene.apache.org/

relacionais, replicação de bases de dados, interface de administração web, etc.[47].

Para o acesso ao Solr, a arquitetura apresentada usa a API SolrJ [47]. O SolrJ é uma API feita em JAVA que permite a comunicação com Solr e oculta detalhes da análise e codificação das mensagens de ida e volta entre a aplicação e Solr. Na Figura 6.3 é apresentada a arquitetura conceitual de Solr. O Solr é disponibilizado em uma aplicação web, onde o usuário pode interagir diretamente com o índice de documentos.

SIREn como foi descrito na seção 4.4.1, do capítulo 4, é uma extensão de Lucene, que da suporte à indexação de coleções de documentos semiestruturados. Ele complementa Lucene com consultas de tipo RDF e uma eficiente indexação de documentos com um número arbitrátivo de metadados [73].

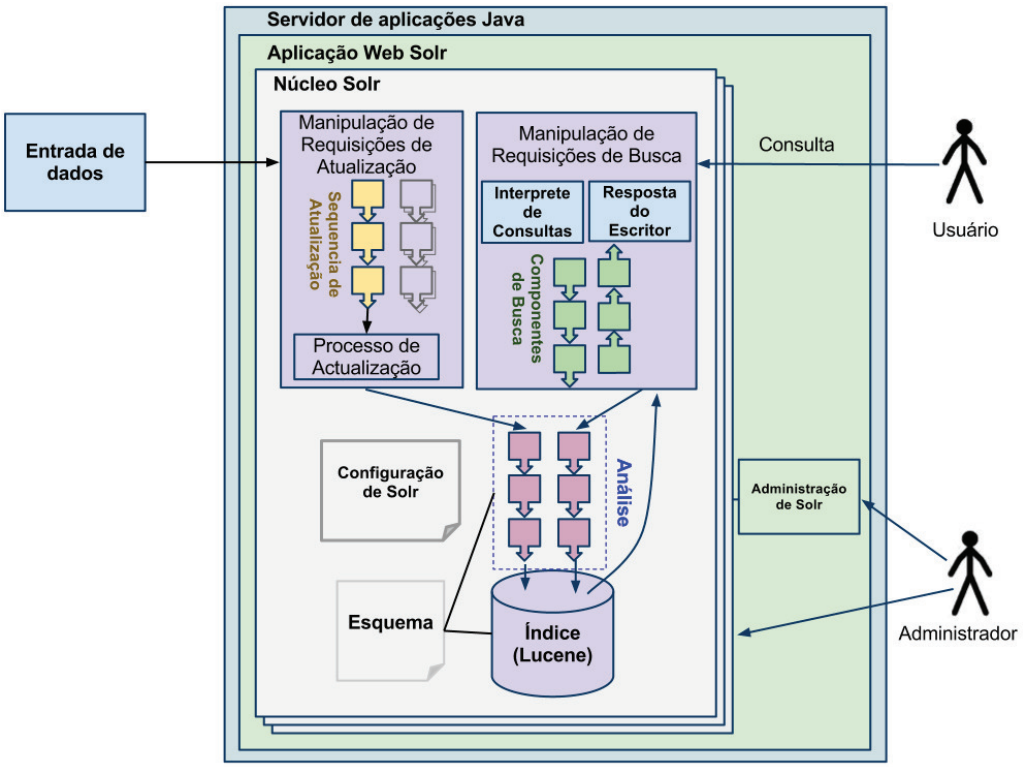

Figura 6.3: Arquitetura conceitual de Solr [74]

5. **Acesso ao índice semântico na Web**: Para o acesso ao índice semântico, uma interface é adaptada para encaminhar as diferentes buscas geradas pelo processo proposto. O índice semântico atualmente usado pela arquitetura é o Sindice.

   Sindice é uma infraestrutura de indexação que disponibiliza os dados através de um *Web front-end* e uma API para encontrar fontes de

dados como arquivos RDF e SPARQL *endpoints* [12]. A API usada para enviar as requisições de busca é Sindice4J[13][75]. A API Sindice4J suporta os diferentes serviços que disponibilizam os dados de Sindice [75]. A Sindice4J pode ser adaptada para suportar diferentes índices semânticos.

É importante ressaltar que SIREn foi desenvolvido para ser o motor de busca de Sindice. Assim, SIREn habilita a indexação de grandes quantidades de dados semiestruturados, consultas e atualizações em tempo real, assim como buscas distribuídas [73].

6.1.2 Componentes Web

A nossa arquitetura consume recursos disponíveis na Web e estes recursos devem ser indicados pelo usuário. Os recursos podem estar disponíveis em fontes de dados RDF, que são representados normalmente por um SPARQL *endpoint* ou arquivos RDF *dump*. Eles são disponibilizados em servidores Web e podem ser acessados usando a URI que os identifica.

A arquitetura de nossa ferramenta considera algumas restrições que uma fonte de dados aplica sobre recursos contidos nela. Restrição como: (i) a quantidade de resultados, no caso de DBPedia é limitada em 10000 e em `data.open.ac.uk` a 200. (ii) O tempo de espera para uma nova requisição é uma restrição que deve ser configurada na interface de acesso, assim podemos melhorar a coleta de dados. (iii) O número de tentativas que uma consulta deve ser executada, isto quando o serviço de dados SPARQL *endpoint* não esteja disponível. Estas considerações afetam o tempo de processamento e nível de efetividade do processo.

O acesso aos recursos contidos no índice semântico é definido também por restrições de acesso. O Sindice limita os resultados de uma busca por páginas. Uma busca pode recuperar 100 páginas, cada página recuperada contém 10 resultados. Em total no Sindice podemos recuperar 1000 resultados. Porém, temos que considerar que o acesso a uma nova página corresponde à uma nova requisição.

6.1.3 Componentes na Nuvem

A arquitetura desenvolvida aproveita recursos na nuvem e estes recursos são disponibilizados por provedores de tecnologia através de diferentes serviços. A *Amazon Web Services* disponibiliza três serviços que são consumidos pela arquitetura: EC2, S3 e EMP. A EC2 disponibiliza recursos de computação, o S3

[13]http://code.google.com/p/sindice4j/

habilita o armazenamento necessário para carregar os dados e o EMP habilita a computação de dados intensivos (do inglês *Data Intensive Computing*). A seguir descreve-se como é usado cada um destes serviços.

1. **Recursos de computação (EC2)**: Como foi definido anteriormente, a arquitetura considera dois tipos de instâncias. No EC2 uma instância é nomeada como *Amazon Machine Image* (AMI) [69]:

   Instância para o acesso RDF *dump*: Esta instância é gerada usando uma AMI que contém como Sistema Operacional Linux e o *software* necessário para executar um servidor Virtuoso[14]. Virtuoso é um banco de dados em triplas [76], que contém todas as capacidades necessárias para carregar documentos RDF e conta com um serviço de SPARQL *endpoint*. Virtuoso fornece suporte a formatos como: rdf, xml, nt, owl ou ttl, e possui uma carga de dados em massa [76].

   Instância para a indexação e a busca de dados: A instância de indexação contém como sistema operacional Linux e o servidor Solr para atender o serviço de indexação e busca.

2. **Armazenamento (S3)**: O serviço de armazenamento oferece suporte à carga de dados do processo proposto, descrito no capítulo anterior e participa ativamente no processamento de dados.

3. **Processamento de dados intensivos (EMP)**: Este serviço executa os processos MapReduce de *Geração de Estatística*, *Escolha de classes* e *Escolha de indivíduos*. Parte destes processos foram descritos no capítulo 4. Na seguinte subseção apresenta-se a modelo MapReduce destes processos.

   No desenvolvimento dos modelos MapReduce deste trabalho é usado o *framework* Hadoop [36]. O Hadoop permite a criação de um ecossistema de negócios baseados em distribuições específicas e o surgimento de serviços em nuvem, como o *Amazon Elastic MapReduce*. O *Amazon Elastic MapReduce* permite tratar dados em massa (nem só as empresas) sem demandar aquisição de servidores físicos. Neste modelo, o usuário desenvolve uma aplicação Hadoop e a roda em cima da nuvem da *Amazon* [36].

[14]http://virtuoso.openlinksw.com/

6.2 Modelo MapReduce do processamento proposto

As etapas de coleta de dados do processo proposto apresentado no capítulo 4 estão ligadas com um modelo MapReduce. É necessário ressaltar que estes modelos são genéricos e podem ser usados independentemente da arquitetura, isto sobre um ambiente Hadoop. A seguir descrevemos os modelos MapReduce da arquitetura definida neste capítulo:

1. **Geração de Estatística**

   O modelo MapReduce da Geração de estatística recebe uma lista de consultas armazenadas no serviço de armazenamento (S3), como escolher as consultas desta lista foi descrito na seção 4.2, no capítulo 4 . Cada consulta é atribuída a um *mapper*. No *mapper* a consulta é executada usando Jena sobre a fonte escolhida pelo usuário e que recupera os resultados em formato de triplas. As triplas são reunidas em documentos do tipo de classes e de tipo estatística como se apresenta nas tabelas B.1 e B.2 respectivamente, no apêndice B . Gerados os documentos, SolrJ envia cada um deles ao serviço de indexação. A Figura 6.4 apresenta o modelo MapReduce da geração de estatísticas.

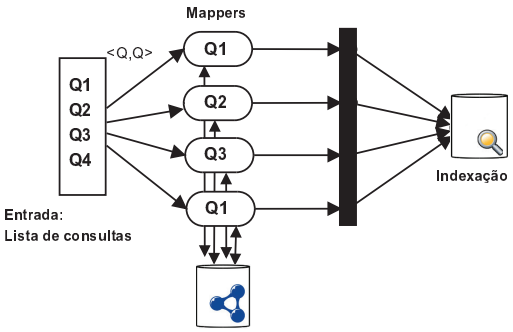

Figura 6.4: Modelo MapReduce da geração de estatísticas

2. **Escolha de classes**

   O modelo MapReduce da escolha de classes recebe uma lista de classes contidas no serviço de armazenamento (S3). O processo de escolha das classes foi descrito na seção 4.5, no capítulo 4. O fluxo de tarefas MapReduce é composto por três passos de execução, como ilustra a Figura 6.5. A seguir descrevemos estes passos:

   (a) Contagem de indivíduos: É necessário realizar uma contagem dos indivíduos que pertencem a uma classe. Como é o caso de DBpedia, onde existem classes que excedem o valor de **fator**, que limita a

quantidade de resultados retornados pela fonte de dados. Sobre cada *mapper* uma consulta **Q** correspondente a uma classe **C** é executada por Jena sobre a fonte de dados escolhida. Se as classes excedem o **fator** da fonte, um conjunto de consultas **Q'** é gerada para extrair os indivíduos da classe **C**, usando a propriedade *offset* de SPARQL. A lista de consultas é salva no serviço de armazenamento através dos *reducers*.

(b) Coleta de indivíduos: Cada consulta **Q'** é atribuída a um *mapper* e a consulta é executa com Jena para extrair os indivíduos da respectiva classe . Os *reducers* filtram os indivíduos duplicados e os envia ao serviço de armazenamento.

(c) Operação *merge*: Neste passo cada indivíduo é atribuído a um *mapper*. Cada *mapper* executa uma operação *merge* como mostra a figura 4.6, detalhada na seção 4.5. Os resultados da operação *merge* são enviados diretamente ao serviço de indexação através de SolrJ, usando a estrutura de documento da tabela B.3.

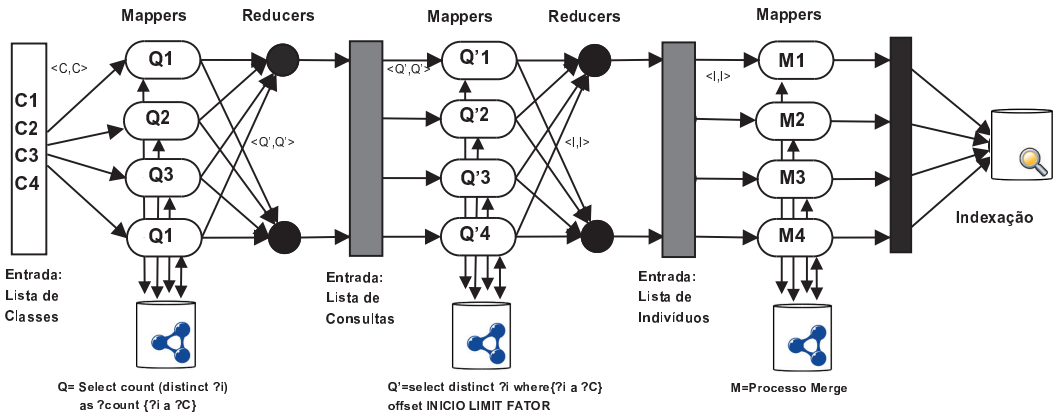

Figura 6.5: Modelo MapReduce da escolha de classes

Por questões de otimização, pode-se atribuir mais de uma operação *merge* a um *mapper*, e assim coletar a informação de vários indivíduos paralelamente. Esta quantidade de operações *merge* é limitada pela capacidade da memória do recurso onde são executados os processos Hadoop.

3. **Escolha de indivíduos**

O modelo MapReduce da escolha de indivíduos recebe uma lista de indivíduos contidos no serviço de armazenamento (S3). A escolha de indivíduos foi descrita no capítulo 4 na seção 4.4. O fluxo de tarefas

MapReduce está composto por dois passos de execução, como ilustra a Figura 6.6. A seguir descrevemos estes passos:

(a) Extração de indivíduos: Cada indivíduo da lista de entrada é atribuído a um *mapper*, em cada *mapper* a API SolrJ envia uma consulta de busca **Q** para o índice que armazena os dados coletados previamente. De cada indivíduo são extraídos as palavras-chave segundo o tipo de busca escolhido. Os tipos de busca são apresentados na Figura 4.8. Cada combinação das palavras-chave representa uma busca **Q'**, assim, uma lista de buscas por cada indivíduo é gerada e enviada ao serviço de armazenamento.

(b) Busca por palavra-chave: Cada busca **Q'** é atribuída a um *mapper*. Em cada *mapper* a API Sindice4J envia a consulta ao índice semântico para extrair os indivíduos da busca. Os resultados são enviados ao serviço de indexação por SolrJ, usando a estrutura de documento da tabela B.3, no apêndice B.

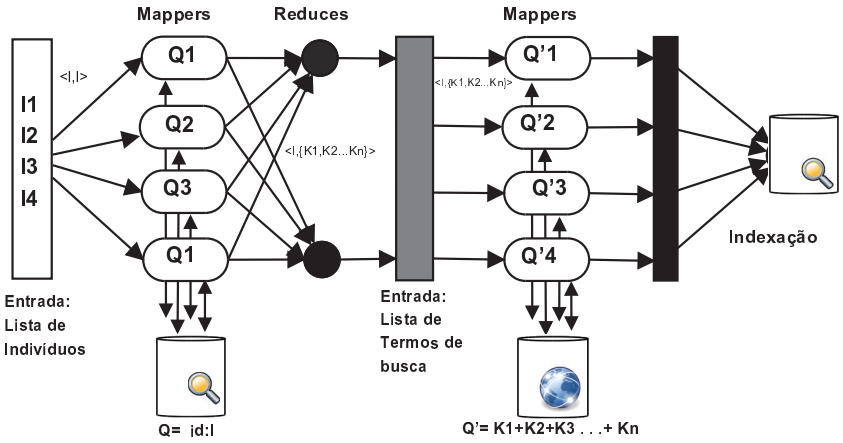

Figura 6.6: Modelo MapReduce da busca de indivíduos

6.3 Opções de otimização

O modelo de MapReduce, definido na seção 2.4, do capítulo 2, requer de um conjunto de configurações para distribuir o processamento entre as diferentes tarefas.

As restrições de cada uma das tecnologias usadas, devem ser sincronizadas junto com as capacidades do modelo MapReduce. A seguir descrevemos as opções de otimização que são necessárias configurar para cada uma das etapas do processo proposto.

Número de computadores: Esta opção é obrigatória em cada tarefa MapReduce e depende dos recursos disponíveis e das limitações das tecnologias usadas. Esta opção aborda também as capacidades do *hardware* usado.

Número de linhas por *mapper*: A principal característica do modelo MapReduce é a divisão do volume da informação que se pretende processar [33]. Esta característica ajuda a distribuir a carga de processamento de cada tarefa MapReduce [33]. Habilitando a opção de atribuir uma quantidade de linhas a cada tarefa *mapper*, consegue-se otimizar o tempo ocioso da cada tarefa. Esta opção de informação está fortemente relacionada como o volume estimado da informação.

Tempo de espera de uma nova requisição: A disponibilidade tanto do serviço de SPARQL *endpoint*, quanto do serviço de indexação de Solr e do Sindice dependem da quantidade máxima de requisições que eles podem atender. Se a opção de definir um tempo de espera for habilitada, caso uma requisição falhe porque o serviço não está disponível, uma nova requisição é enviada ao respectivo serviço.

Tentativas por requisição: Esta opção está diretamente relacionada com o tempo de espera de uma nova requisição. Esta opção define a quantidade de vezes que pode ser enviada uma requisição.

Número de processos *merge*: Na etapa de escolha de classes. na seção 4.4.1, é necessária uma operação *merge* para extração de informações de cada indivíduo. Define-se a opção de atribuir a um *mapper* a execução de mais uma operação *merge* paralelamente, para otimizar o tempo de processamento. Porém, esta opção acelere o tempo de processamento executando múltiplas operações *merge*, deve-se considerar a capacidade de memória disponível nos computadores responsáveis do modelo MapReduce. Assim, para definir um número adequado de operações *merge* deve-se ter conhecimentos das capacidades dos recursos usados.

No processamento de uma operação *merge* é necessário 50 *megabytes* de memória. Para estimar de forma mais precisa, os recursos do sistema operacional e software fundamental para o processamento devem ser considerados. Em um computador de 15 *gigabytes*, com um total de 8 *mappers* rodando paralelamente e com 10 operações *merge* por *mapper*, apenas 4 *gigabytes* poderiam ser aproveitados para executar operações *merge*. Em outras palavras, precisa-se de 500 *megabytes* por cada *mapper*

além dos recursos necessários que o modelo MapReduce precisa para rodar.

Número de termos de busca: Na etapa de escolha de indivíduos, na seção4.4, é necessário um processo de extração de palavras-chave para fazer a busca sobre índice semântico. O número de palavras-chave para executar a busca sobre Sindice é definido para limitar o número de buscas e consequentemente limitar o processamento de dados.

Número de documentos indexados: A quantidade de documentos enviados ao serviço de indexação afeta a disponibilidade do serviço e o tempo de processamento. Esta tarefa é assumida pelo Solr, mas deve ser configurada segundo a quantidade das tarefas MapReduce possíveis para uma indexação aceitável.