MAPREDUCE KULLANARAK RDFS ÜZERİNDE DAĞITIK ÇIKARSAMA

YİĞİT ÇETİN

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ

TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

TEMMUZ 2014

ANKARA

Fen Bilimleri Enstitü onayı

_____

Prof. Dr. Osman EROĞUL

Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

_____

Doç. Dr. Erdoğan DOĞDU

Anabilim Dalı Başkanı

Yiğit ÇETİN tarafından hazırlanan MAPREDUCE KULLANARAK RDFS ÜZERİNDE DAĞITIK ÇIKARSAMA adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

_____

Doç. Dr. Osman Abul

Tez Danışmanı

Tez Jüri Üyeleri

Başkan : Doç. Dr. Erdoğan DOĞDU _____

Üye : Doç. Dr. Osman ABUL _____

Üye : Doç. Dr. Pınar KARAGÖZ _____

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada orijinal olmayan her türlü kaynağa eksiksiz atıf yapıldığını bildiririm.

Yiğit ÇETİN

Üniversitesi : TOBB Ekonomi ve Teknoloji Üniversitesi

Enstitüsü : Fen Bilimleri

Anabilim Dalı : Bilgisayar Mühendisliği

Tez Danışmanı : Doç. Dr. Osman Abul

Tez Türü ve Tarihi : Yüksek Lisans – Temmuz 2014

Yiğit ÇETİN

MAPREDUCE KULLANARAK RDFS ÜZERİNDE DAĞITIK ÇIKARSAMA

ÖZET

Günümüzde teknoloji çağında büyük veriler üzerinde işlemlerin gerçekleştirilebilmesi gerekmektedir. Bu verilerin ölçeklenebilir bir şekilde gerçekleştirilebilmesi için en önemli çözüm yolu dağıtık ve paralel hesaplama yöntemidir. MapReduce, büyük veri kümeleri üzerinde ve çok sayıda bilgisayarı birlikte kullanarak gerekli işlemleri dağıtık ve paralel olarak gerçekleştiren bir programlama modelidir. Öte yandan RDFS veri kümelerinin dosya boyutu ve karmaşıklığı giderek artmaktadır. Bu da RDFS çıkarsama işlemlerinde performans sıkıntılarına neden olmaktadır. Bunun çözümü için büyük veri çözümleri kullanılabilir. RDFS çıkarsama sonucu oluşan çıktılar başka bir çıkarsama işlemi için girdi olabilir böylece eldeki büyük veriler için çıkarsama yapılırken veriler giderek büyür. Bu çalışmada Hadoop üzerinde kullanılan paralel ve dağıtık çalıştırma teknolojileri MapReduce, Hive ve Pig kullanarak dağıtık RDFS çıkarsama işlemi gerçekleştirip, performanslarını gözlemleyeceğiz. Ayrıca doküman indeksleme yöntemi kullanarak performans artışı sağlamayı amaçlamaktayız. Bunun içinde MapReduce ve indeksleme yaklaşımları kullanılarak efektif ve dikkate değer bir şekilde çıkarsama işleminin gerçekleştirebileceği gösterilmektedir. Deneysel çalışmalar Dbpedia ve Freebase veri kümeleri üzerinde gerçekleştirilmiştir.

Anahtar Kelimeler: Hadoop, MapReduce, RDFS Çıkarsama, Semantik Ağ

University : **TOBB Economics and Technology University**
Institute : **Institute of Natural and Applied Sciences**
Science Programme : **Computer Engineering**
Supervisor : **Associate Professor Dr. Osman Abul**
Degree Awarded and Date : **M.Sc. –2014 July**

Yiğit ÇETİN

IMPLEMENTATION OF DISTRIBUTED RDFS REASONING WITH MAPREDUCE

ABSTRACT

We live in big data age in which many computational tasks either generate or need to use large datasets. This makes parallel and distributed computing a key for scalability. MapReduce is a programming model for processing large datasets in parallel and distributed fashion on cluster of computers. Today, since the size and complexity of RDFS documents increase rapidly, RDFS reasoning problem has to embrace and address the big data solutions. The output of RDFS reasoning job can be input to another job and the output of RDFS reasoning jobs grow big as the input documents gets bigger. In this study, MapReduce programming model, in particular Hadoop with related core technology like Hive and Pig, is used for improving the performance of distributed RDFS reasoning. Additionally, document indexing is used for further performance gain. The study shows that both of the MapReduce and indexing approaches are quite effective and offer significant performance improvements and scalable solutions. Experimental evaluations on two RDFS datasets, namely Dbpedia and Freebase, are provided.

Keywords: Hadoop, MapReduce, RDFS Reasoning, Semantic Web

TEŞEKKÜR

Öncelikle bu çalışmayı birlikte yürüttüğümüz, her adımdan desteğini esirgemeyen danışman hocam Osman Abul'a, iş ortamında Hadoop konusunda çalışmama yön veren ve çalışmalarımı destekleyen yöneticim Ali Toksoy'a çok teşekkür ederim.

TOBB Ekonomi ve Teknoloji üniversitesinde gelişimime katkıda bulunan tüm hocalarıma, Lisans döneminde akademik hayata da ilgi duymama vesile olan Övünç Çetin ve Erdem Eser Ekinci'ye, yine lisans döneminde gelişimime katkıda bulunan başta Prof. Dr. Oğuz Dikenelli olmak üzere tüm Ege Üniversitesi Bilgisayar Mühendisliğindeki hocalarıma teşekkür ederim.

Son olarak zor zamanlarımda beni destekleyen dostlarıma ve her zaman yanımda olan aileme de sonsuz teşekkürler ederim.

İÇİNDEKİLER

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

KISALTMALAR

Kısaltmalar	Açıklama
HDFS	Hadoop Distributed File System
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
IO	Input / Output
CPU	Central Processing Unit
VM	Virtual Machine
W3C	World Wide Web Consortium

1. GİRİŞ

Semantik Web kavramı ilk olarak Tim Beerners-Lee, James Handler ve Ora Lassila tarafından ortaya atılmıştır. Semantik web, yazılan bilgilerin sadece doğal diller ile değil, yazılımlar tarafından yorumlanıp, kullanılabilecek şekilde tanımlanabilmesini sağlayan bir internet teknolojisidir [4]. Amacı internet ortamında bulunan bilgilerin sadece kullanıcılar tarafından değil yazılımlar tarafından da anlaşılabilmesi ve bu özellikleri kullanarak bazı işlemlerin otomatikleştirilebilmesidir.

Bu bilgilerin tanımlanabilmesi için ortak bir standart belirlenmesi gerekmektedir. Bunun içinde RDF dili üretilmiş ve standart olarak belirlenmiştir. RDF dili zamanla bazı ihtiyaçları karşılayamaz duruma gelmiştir. Bu sebeple de bu dile yapılan eklemeler ile RDFS ortaya çıkmıştır. RDFS'de RDF'den farklı olarak ontolojilerdeki kaynaklar arasında ilişki oluşturmak için yeni standartlaştırılmış ve özel anlamlı tanımlamalar eklenmiştir. Bu tanımların eklenmesiyle de kaynaklar arasında birçok ilişki tanımlanabilmektedir. Yani RDFS ile kaynaklar arasında bağlantılar tanımlanabilmekte ve ilgili kaynaklar bu şekilde anlamsal olarak birlikte değerlendirilebilmektedir. Ayrıca eldeki tanımlamalardan yeni tanımlamalar ortaya çıkarma durumu da ortaya çıkmaktadır. Bu da RDFS üzerinden çıkarsama yapabilmemizi sağlamaktadır.

RDFS çıkarsama işlemlerinin gerçekleştirilmesi önemli bir IO ve CPU gücü gerektirebilmektedir. Özellikle büyük veri kümeleri üzerinde çıkarsama işlemi gerçekleştirilmek istendiğinde bu işlem uzun sürebilir. Bu darboğazında üstesinden gelebilmek için paralel hesaplama teknolojileri kullanılabilir [6].

Hadoop [7] büyük verileri işlemek için açık kaynak olarak gerçekleştirilen ve çok sayıda bilgisayarın bir arada dağıtık olarak çalışmasını sağlayan güvenilir ve ölçeklenebilir bir yapıdır. Hadoop sağladığı dağıtık dosya sistemi ve desteklediği paralel işlem gücü ile önemli performans artışı sağlamaktadır. Hadoop 4 adet alt parçadan oluşmaktadır:

- Hadoop Genel
- Hadoop Dağıtık Dosya Sistemi
- Hadoop Yarn
- Hadoop MapReduce

Hadoop Genel'de diğer parçalar tarafından kullanılacak olan kütüphaneler ve hizmetleri sağlamaktadır. Dağıtık dosya sistemi, birlikte çalışan makinaları bir arada kullanarak yüksek ölçeklenebilirlik ve güvenilirlikle dosyaların saklanabilmesini sağlamaktadır. Hadoop Yarn kaynakların yönetiminden sorumludur. Son olarak Hadoop MapReduce büyük verilerin işlenmesi için bir programlama modelidir.

MapReduce fonksiyonel programlamadaki map ve Reduce fonksiyonlarından esinlenerek üretilmiş büyük verilerin paralel ve dağıtık bir şekilde işlenmesini sağlayan bir programlama modelidir. İki aşamadan oluşmaktadır: Map ve Reduce aşamaları. Map aşamasında gelen girdiler gerekli işlemlerden geçirilip bir ara sonuç oluşturulur. Bu sonuçlarda Reduce fonksiyonuna girdi olur ve bu girdiler üzerinden yapılan işlemler ile de sonuç değeri oluşturur.

MapReduce fonksiyonununda girdiler <anahtar,değer> şeklindedir ve bu girdilerde fonksiyonlara anahtar sahasının değerine göre dağıtılıp işlenir. Map aşamasındaki her bir görev birbirinden bağımsızdır. Yani birbirilerinin çalışmasını etkilemezler. Bu sebepten ötürü de paralel olarak çalıştırılabilmektedirler. Aynı şekilde reduce aşamasındaki her bir görev birbirinden bağımsız ve paralel olarak çalıştırılabilmektedir.

Hadoop MapReduce kodlanması Java ile gerçekleştirilmektedir. Bu sebepten ötürü programlama bilgisine sahip olmayan kişilerin MapReduce ile analiz işlemlerini gerçekleştirebilmesi oldukça güç olmaktadır. Bu nedenden ötürü Hadoop ve MapReduce ile çalışan bazı araçlar gerçekleştirilmiştir. Bunlardan çok kullanılanları Facebook tarafından geliştirilen Hive[3] ve Yahoo tarafından geliştirilen Pig'dir[9].

Hive SQL bilgisine sahip kişilerin Hadoop dosya sistemi üzerinde saklanan veriler üzerinde analiz işlemi gerçekleştirebilmesi için geliştirilmiştir. Burada veriler

Hive ile Çizelgeler halinde sistemde saklanıyormuş gibi görünmekte ve son kullanıcı SQL benzeri bir dil (HQL) ile sorgulama işlemlerini gerçekleştirmektedir. Yazılan bu sorgu MapReduce görevine çevrilmekte ve sistemde bu şekilde çalıştırılmaktadır.

Pig ise PigLatin adı verilen bir betik dili ile Hadoop dosya sistemi üzerinde saklanan verilerde analiz işlemini gerçekleştirebilmektedir. Hive'e benzer şekilde PigLatin koduyla yazılmış işlemleri MapReduce görevine çevirmekte ve dosya sistemi üzerinde çalıştırmaktadır.

Sonuç olarak RDFS çıkarsama işleminde gerekli olan disk IO ve CPU performansı kısıtlarının üstesinden gelmek için bulut teknolojileri [12] ve Hadoop ile onun sağladığı MapReduce görevleri kullanılabilir. Bu tez kapsamında da Hadoop ve onun sağladığı diğer yapılar kullanılarak RDFS çıkarsama işleminin hızlı ve doğru bir şekilde yapılabilmesi amaçlanmaktadır.

Tezin bundan sonraki kısmına bakacak olursak Bölüm 2'de bu tez kapsamında kullanılan altyapı bileşenleri teknik olarak incelenip tanımlanmıştır. Bölüm 3'de RDFS çıkarsama işleminin MapReduce ile nasıl gerçekleştirildiği ve ne gibi yöntemler kullanılabileceği anlatılmıştır. Bölüm 4'te Hive ile RDFS çıkarsama işleminin gerçekleştirimi anlatılmıştır. Bölüm 5'de Pig ile RDFS çıkarsama işleminin nasıl gerçekleştirildiği anlatılmıştır. Bölüm 6'da daha önceki bölümlerde anlatılan yöntemlerin performans karşılaştırılması yapılmıştır. Son olarak Bölüm 7'de elde edilen sonuçlar genel olarak değerlendirilmiştir.

2. TEKNİK ALTYAPI

Bu bölümde tez kapsamında kullandığımız teknolojiler hakkında bilgilendirmede bulunacağız. Tezin anlaşılabilmesi için bu teknolojiler hakkında bilgi sahibi olunması gerekmektedir.

2.1. Çıkarsama (Reasoning)

Çıkarsama elde bulunan bilgiler ışığında belli kurallara sadık kalınarak eldeki verilerden yeni veriler türetilmesi işlemidir. İki tür çıkarsama tipi bulunmaktadır: Tümdengelim ve Tümevarım.

Bu tez kapsamında biz tümdengelim tipindeki çıkarsama işlemlerini ele alacağız. Tümdengelim çıkarsamada belli terimler doğruysa belli sonuçların da doğru olması gerekmektedir. RDF dilinde bir örnek ile inceleyecek olursak:

Yiğit isa Student .

Student subclassof Person

Eğer Yiğit öğrenciyse ve öğrenci de aynı zamanda insan ise burada tümevarım çıkarsama sonucu olarak Yiğit'in insan olduğu çıkarılabilir.

Yiğit isa Person

Çıkarsama ileri dönük çıkarsama ve geriye dönük çıkarsama şeklinde de ikiye ayrılır. Bu ayrımın girdinin sistemdeki başlangıç noktasına göre belirlenmektedir.

Eğer eldeki girdileri başlangıç noktası olarak alıyor ve bu girdilere göre oluşabilecek tüm çıktıları oluşturuyorsak bu işlem ileri dönük çıkarsama işlemidir. Yukarıdaki örnek ileri dönük çıkarsamaya örnektir. Burada 2 üçlü değeri girdi olarak alınmış ve bunun sonucunda 1 adet çıktı oluşturulmuştur.

Geriye dönük çıkarsama da ise beklenen sonuç ele alınır (Örneğin **Yiğit isa Person**). Geriye dönük özyinele kullanılarak üçlüler oluşturulur. Eğer girdideki üçlüler oluşturulabilirse geriye dönük çıkarsama işlemi gerçekleştirilmiş olur.

İki çıkarsama da farklı alanlarda kullanılmaktadır. İleri dönük çıkarsama eldeki girdilerle yeni üçlüler oluşturulmak istendiğinde kullanılmaktadır. Geriye dönük çıkarsama ise oluşan sonuçların doğruluğunu kontrol etmek amacıyla bir sorgu gibi kullanılmaktadır.

Bu tez kapsamında biz ileriye dönük çıkarsama işlemini kullanacağız ve eldeki girdilerden elde edilecek tüm sonuçları oluşturuncaya kadar çıkarsama işlemi yapmaya devam edeceğiz.

2.2. MapReduce

MapReduce Google tarafından geliştirilmiş olan dağıtık bir programlama modelidir. Büyük veri kümelerini işlemek ve büyük veri kümelerini performanslı bir şekilde oluşturmak için kullanılır.

Fonksiyonel programlamadan esinlenerek oluşturulmuştur. Fonksiyonel programlamada da bulunan iki fonksiyon olan map() ve reduce() kullanılarak gerçekleştirilmek istenen işlemler gerçekleştirilmektedir. Burada tüm girdiler <key,value> yapısı şeklindedir. Öncelikle girdiler map fonksiyonuna girer. Map fonksiyonu gerekli işlemleri yaparak ara bir "anahtar-değer" (intermediate value) üretir. Bu veriler anahtar değerlerine göre gruplanarak reduce fonksiyonuna girdi olur. Aynı şekilde reduce fonksiyonunda da gerçekleştirilmek istenen işlemler yapılır ve anahtar-değer yapısındaki sonuç değeri oluşturulur.

Map fonksiyonun örnek ile açıklamak gerekirse:

Aşağıdaki map fonksiyonunda anahtar ve değer yapısındaki girdileri alıp aynı formatta verilerin büyük harfli sürümlerini üretmektedir.

- let map(k, v) = emit(k.toUpper(), v.toUpper())

- (“foo”, “bar”) -> (“FOO”, “BAR”)
- (“Foo”, “other”) -> (“FOO”, “OTHER”)
- (“key2”, “data”) -> (“KEY2”, “DATA”)

Reduce fonksiyonun örnek ile açıklamak gerekirse:

Aşağıdaki anahtar değer şeklinde girilen girdi değeri vektördeki her değeri dolaşarak aynı anahtar ile eşleştirilmektedir.

- let reduce(k, vals) = foreach v in vals: emit(k, v)

(“A”, [42, 100, 312]) -> (“A”, 42), (“A”, 100), (“A”, 312)

MapReduce programlama modeli sağladığı yapı sayesinde işlemlerin paralel gerçekleştirilmesini desteklemektedir. Bu işlemlerin dosyanın saklandığı yerde gerçekleştirilmesi amaçlanmaktadır. Böylece ağ üzerinde dosyaların taşınması sırasında zaman kaybının önüne geçilebilmektedir.

Map fonksiyonları birbirinden tamamen bağımsız olarak çalışmaktadır yani hepsi aynı anda paralel olarak çalıştırılabilir. Ancak gerçek kullanımda elde bulunan kaynakları(CPU, RAM vb.) paylaştıklarından ötürü sunucunun kalitesine göre aynı anda belli sayıda paralel map görevi çalıştırılmaktadır. Diğer map görevleri çalışan map görevleri bittikçe çalışmaya başlamaktadır. Bu durum reduce görevleri içinde geçerlidir.

MapReduce işlemini sırasında performans kaybı yaşamamak için girdi ve çıktıların dağıtık bir dosya sistemi üzerinde saklanması gerekmektedir. Çünkü normal dosya sistemi üzerinde çalışan MapReduce paralelleştirmeyi tam olarak gerçekleştirmediği için bir avantaj sağlayamamaktadır.

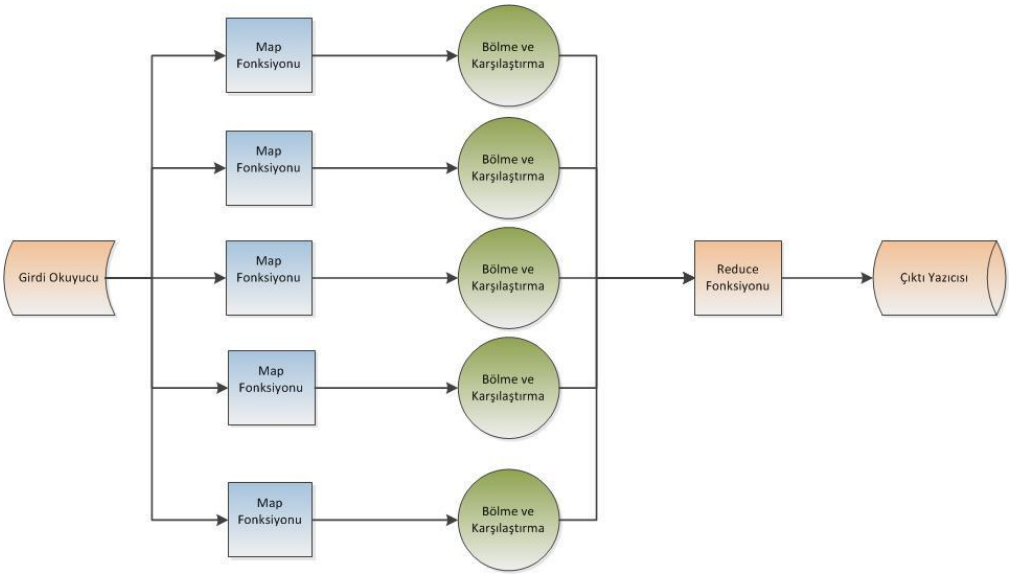
MapReduce işleminin iş akışını inceleyecek olursak altı aşamadan oluştuğunu görüyoruz:

- Girdi Okuyucu
- Map Fonksiyonu

- Bölme Fonksiyonu
- Karşılaştırma Fonksiyonu
- Reduce Fonksiyonu
- Çıktı Yazıcısı

Bu aşamalar arasındaki ilişki aşağıdaki şekilde belirtilmektedir.

Birinci aşama olan girdi okuyucu aşamasında girdi belli büyüklükte parçalara bölünür ve her bir parça için bir map fonksiyonu çalıştırılır. Ayrıca girdi okunurken de map fonksiyonun girdi şekli olan <anahtar,değer > yapısına bu aşamada çevrilir. Bu aşama girdinin map fonksiyonuna uygun hale getirildiği aşamadır.



Şekil 2.1. Map Reduce İş Akışı

Map fonksiyonuna <anahtar,değer > şeklindeki listeler girdi olarak alınır ve yapılmak istenen işlem sonucunda ara bir <anahtar,değer > sonucu üretilir ancak map fonksiyonun uygun girdilere sahip değilse herhangi bir sonuç oluşturmadan da son bulabilir. Map fonksiyonun girdi ve çıktı tipi birbirinden farklı olabilir. Yani map fonksiyonu string tipinde bir değeri girdi olarak alıp sonucunda integer tipinde bir değer oluşturabilir.

Bölme fonksiyonunda kaç tane reducer fonksiyonunun çalıştırılacağı ve map fonksiyonu sonucunda oluşan çıktı değerlerinin hangi reducer fonksiyonu girdi

olacağına karar verilmektedir. Bu aşama da “shuffle” olarak adlandırılan işlemde gerçekleştirilmektedir. "Shuffle" işlemi map fonksiyonu sonucunda oluşan değerlerin reduce işleminin gerçekleştireceği makinaya taşınması işlemidir. Bu taşıma işlemi IO hızı ve ağ hızına göre değişiklik göstermektedir.

Karşılaştırma fonksiyonunda map fonksiyonundan gelen girdilerin sıralama işlemi gerçekleştirilmektedir.

Reduce aşamasında sıralı olarak gelen <anahtar,değer> yapısı reduce fonksiyonunda çalıştırılır ve sonuç değeri oluşturulur.

Son olarak çıktı yazıcısı aşamasında oluşan sonuç değerlerinin dosya sistemine yazılması işlemi gerçekleştirilmektedir.

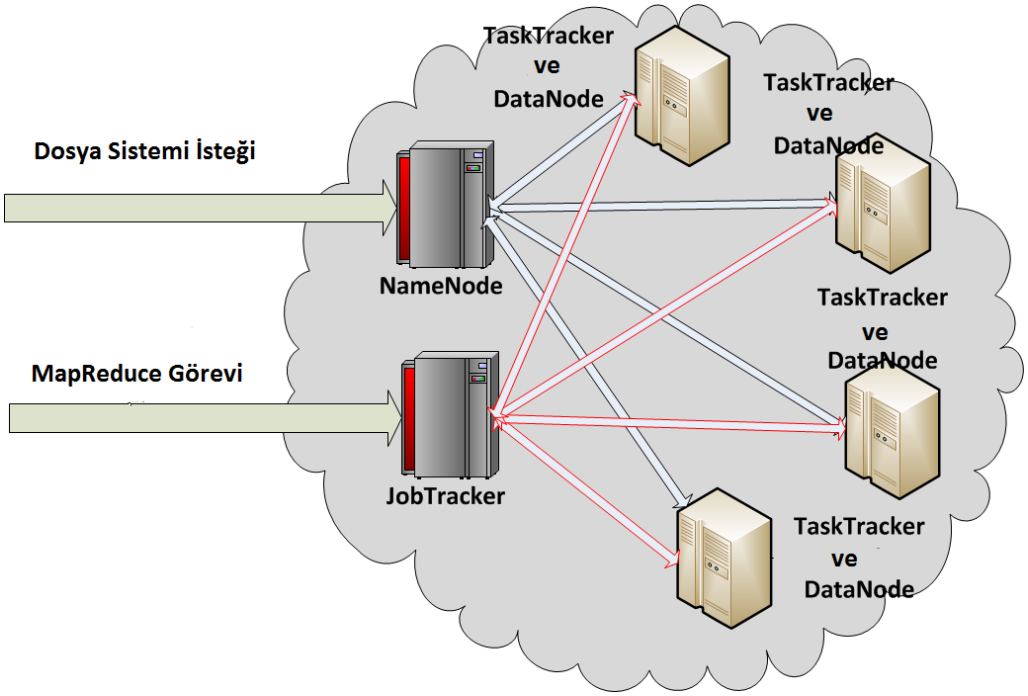
2.3. Hadoop Programlama Çatısı

Hadoop basit programlama modelleri kullanarak büyük veri kümeleri üzerinde işlemlerin gerçekleştirmesine izin veren bir programlama çatısıdır [5][14]. Bir makinadan binlerce makinaya kadar ölçeklenebilecek şekilde tasarlanmıştır. Burada her makine kendi hesaplama ve depolama birimlerini kullanmaktadır. Karşılaşılan hatalar donanım tarafından tespit edilmek yerine uygulama üzerinde tespit edilip çözüme kavuşturulacak şekilde bir yapı tasarlanmıştır. Yani bir makinada karşılaşılan problem o makina tarafından değil Hadoop uygulaması tarafından ele alınıp çözüme kavuşturulmakta aksi takdirde o makina sistemden çıkarılmaktadır.

Aşağıdaki şekilde de görülebileceği gibi Hadoop iki ana yapıdan oluşmaktadır: MapReduce ve Dağıtık Dosya Sistemi.

Hadoop sistemi kendi dosya sistemine sahiptir(HDFS). Bu dosya sistemi dağıtık olarak çalışacak şekilde tasarlanmıştır. Bu dosya sisteminin yönetimi Namenode adı verilen Hadoop yapısı tarafından sağlanmaktadır. Cluster içindeki makinalardan biri Namenode olarak atanır ve bu makine Hadoop dosya sisteminin yönetiminden, hata ayıklamasından sorumludur. HDFS dosya sistemindeki diğer

makinalara da Datanode adı verilmektedir. Datanode olan makinalar dosyaları saklamaktan sorumludur.



Şekil 2.2. Hadoop Çatısı Genel Görünüm

MapReduce tarafının yönetimi de dosya sistemine benzerdir. Burada JobTracker bulut üzerinde çalışacak görevlerin yönetiminden sorumludur. Çalıştırılmak istenen görev JobTracker tarafından ele alınır ve görevin büyüklüğüne göre parçalara ayrılıp TaskTracker'lara aktarılır. TaskTrackerlar kendilerine atanan görevleri alıp çalıştırmaktan sorumludurlar. TaskTracklerların oluşturduğu sonuçlar daha sonra birleştirilerek MapReduce işleminin sonucu üretilmiş olur.

HDFS dosya sisteminde replikasyonlu bir depolama yapılmaktadır. Böylece bir diskte karşılaşılan bir sıkıntı sistemin çalışmasını direk olarak etkilememektedir. Replikasyon yapısının yönetiminde de Namenode sorumludur. Hangi verinin hangi diskin neresinde saklandığı Namenode tarafından bilinmektedir ve bu bilgiler Namenode makinasının RAM'inde saklanmaktadır. Bu sebepten ötürü bulutun büyüklüğüne göre bu makinanın RAM'in büyüklüğü olabildiğince yüksek düzeyde tutulmalıdır. HDFS dosya sisteminin varsayılan blok büyüklüğü 128 MB'tır. Blok büyüklüğünün bu kadar çok olmasının sebebi dosya sisteminin büyük dosyaları

saklamak ve işlemek üzerine tasarlanmasıdır. HDFS dosya sistemin bir kere yaz çok kere oku işlemlerinde başarılıdır. Yani yazma hızı düşük iken okuma hızında önemli bir performans artışı ortaya koymaktadır. Yazma hızının düşük olmasının sebebi replikasyonlu bir şekilde yazma işleminin gerçekleştirmesidir.

2.4. RDF (Resource Description Framework)

RDF W3C tarafından ortaya atılmış bir veri modelidir. Bu veri modeli 3'lüler şeklinde tanımlanmaktadır. Her üçlü *subject object ve predicateden* oluşmaktadır. Bu üçlülerin oluşturulma biçimi İngilizce söz dizimine benzerlik göstermektedir. Bir örnek ile açıklayacak olursak "Yigit plays football". Burada işi yapan Yigit subject, plays predicate ve football object olmaktadır.

RDF yapıları birden çok formatta gösterilebilmektedir. Çok kullanılan formatlar: RDF/XML, N3, N-triples and Turtle. Birinci format direk olarak XML formatıdır. Bu format XML yapısının sağladığı avantajları desteklemektedir. Bu sebepten yazılımsal olarak RDF üzerindeki işlemleri gerçekleştirmek için bazı durumlarda kolaylık sağlamaktadır. Öte yandan N3 ve N-triples formatları da XML formatından tamamen farklı düz metin formatındadırlar. Bunlarda ise üçlüleri tek satırda arka arkaya subject predicate object sırasında gözlemleyebilmekteyiz. Aşağıda bu formatların örnekleri verilmektedir.

Çizelge 2.1. RDF Format Örnekleri

RDF/XML:

```
<rdf:Description rdf:about="http://www.tobb.etu/student">
<hasName>Yiğit Çetin </hasName>
</rdf:Description>
```

N-Triples:

```
< http://www.tobb.etu/student >
<hasName>
"Yiğit Çetin" .
```

RDF veriler için bize standart bir gösterim sağlamaktadır. Ancak verilerin sadece gösterilmesi bir işe yaramamaktadır. Bu verilerden anlam çıkarılabildiğinde bu gösterim durumu yararlı bir hale dönüşmektedir. RDF verileri üzerinde analiz yapmak içinde SPARQL[10] adında standart bir sorgulama dili kullanılmaktadır. SPARQL diğer sorgu dilleriyle benzerlik göstermektedir. Bu dillerin üçlüler üzerinde sorgulama işlemi gerçekleştirecek şekilde düzenlenmiş halidir.

2.5. RDF Şema (RDFS)

RDF dili için tanımlanmış bir şema dilidir. RDF diline bazı eklemeler yapılarak geliştirilmiş halidir. Ancak bu işlem normal programlama dillerine direk olarak ekleme yapmaktan farklıdır. Çünkü RDF dili zaten yine kendimizin de eklemeler yapabileceği bir yapı sağlamaktadır. Burada dile özel anlama gelen yapılar eklenmekte ve bu yapılar bu dilde standartlaştırılmaktadır [8]. RDFS'de bu benzersiz anlama sahip yeni yapılarla dil güçlendirilmiştir. Bir örnek ile açıklayacak olursak:

Çizelge 2.2. RDFS Örneği

Person rdfs:subClassOf LivingCreature . Yiğit rdf:type Person.

İlk tanımlamada özel bir tanım olan "rdfs:subClassOf" tanımı kullanılmıştır. Bu tanım alt küme ilişkisi tanımlamaktadır. İkinci tanımlamada "rdf:type" özel tanımı kullanılmıştır. Bu tanımda tip tanımlama işlemi için kullanılmaktadır. Bu tanımlar tüm kümelerde aynı anlama gelmektedir ve etki alanından bağımsızdır. Bu sebepten bu tanımlamalara sahip kümeler arasında ilişki kurulabilir. Bu tez kapsamında da bu özel tanımlar kullanılarak çıkarsama işlemleri gerçekleştirilecektir.

2.6. RDFS Çıkarsaması

Yukarıdaki örnekteki tanımlamalara bakarsak Yiğit'in yaşayan bir canlı olduğu çıkarsamasında bulunabiliriz. Bunun anlamı eldeki üçlülerden yeni bir üçlü tanımlanmasıdır ve bu işleme çıkarsama denmektedir.

Çizelge 2.3. RDF Tip Örneği

Yiğit rdf:type LivingCreature.

Bu çıkarsama işlemi sürekli bir şekilde girdilere uygulanmaktadır. RDFS'de bu çıkarsama işlemini gerçekleştirebileceğimiz 13 adet kural bulunmaktadır [1]. Bu kurallar aşağıdaki Çizelgede tanımlanmıştır.

Çizelge 2.4. RDFS Çıkarsama Kuralları

Numara	If	Then.
1	s p o (where o is a literal)	n rdf:type rdfs:Literal
2	s rdfs:domain x u s y	u rdf:type x
3	p rdfds:range o s p v	v rdf:type o
4a	s p o	o rdf:type rdfs:Resource
4b	s p o	s rdf:type rdfs:Resource
5	p rdfs:subPropertyOf p1 p1 rdfs:subPropertyOf p2	p rdfs:subPropertyOf p2
6	p rdf:type rdf:Property	p rdfs:subPropertyOf p
7	s p o p rdfs:subPropertyOf p1	s p1 o
8	s rdf:type rdfs:subClassOf	s rdfs:subClassOf rdfs:Resource
9	c rdfs:subClassOf c1 v rdf:type c1	v rdf:type c
10	u rdf:type rdfs:Class	u rdfs:subClassOf u
11	c rdfs:subClassOf c1 c rdfs:subClassof c2	c1 rdfs:subClassOf c2
12	s rdf:type rdfs:ContainerMembershipProperty	s rdfs:subPropertyOf rdfs:member
13	s rdf:type rdfs:Datatype	s rdfs:subClassOf rdfs:Literal

2.7. SOLR

Solr [2] açık kaynak olarak geliştirilmiş olan bir arama motorudur. Lucene [2] projesinin üzerine geliştirilmiş ve ek özellikler kazandırılmış bir arama sunucusudur. Başlıca özellikleri tam metin arama, aranan kelimeleri bulundukları yeri vurgulama ve çok sayıda doküman tipini destekleyebilmesidir. Ayrıca dağıtık arama ve indekslerin kopyalı bir şekilde saklanmasını da sağlamaktadır. Solr ölçeklenebilir bir yapı sağlar ayrıca NoSql veritabanı özelliklerine de sahiptir.

Solr Java dili kullanılarak gerçekleştirilmiştir. Arama işlemleri için Lucene arama motorunu kullanmaktadır. Gerekli ayarlamalar için xml dosyaları kullanılmaktadır. Solr'da iki önemli konfigürasyon dosyası solrconfing.xml ve schema.xml dosyaları bulunmaktadır. Solrconfig.xml dosyasında Solr'ın genel çalışmasıyla ilgili ayarlamalar yapılmaktadır. Schema.xml dosyasında ise indekslenmek istenen verilerin içeriği ve nasıl indeksleneceğiyle ilgili ayarlamalar yapılabilmektedir. Yani bir sahanın tipi, hangi karakterlere göre indekslenebileceği (boşluk, cümle yapısı vb.) bu dosyadaki ayarlamalar ile tanımlanmaktadır.

Gerekli ayarlamalar yapıldıktan sonra Solr sunucusuna Java API ile ulaşılarak indekslenmek istenen değerler sunucuya gönderilmektedir. Sunucu bu istekleri işleyip sistemde saklamaktadır. Solr bu saklama işlemi için sunucunun üzerinde çalıştığı dosya sistemini kullanmaktadır. Burada ilgili klasör içinde oluşturduğu indeks dosyaları içinde ilgili bilgileri saklamaktadır. Solr üzerinde indeksleme işlemi MapReduce görevi ile de yapılabilmektedir.

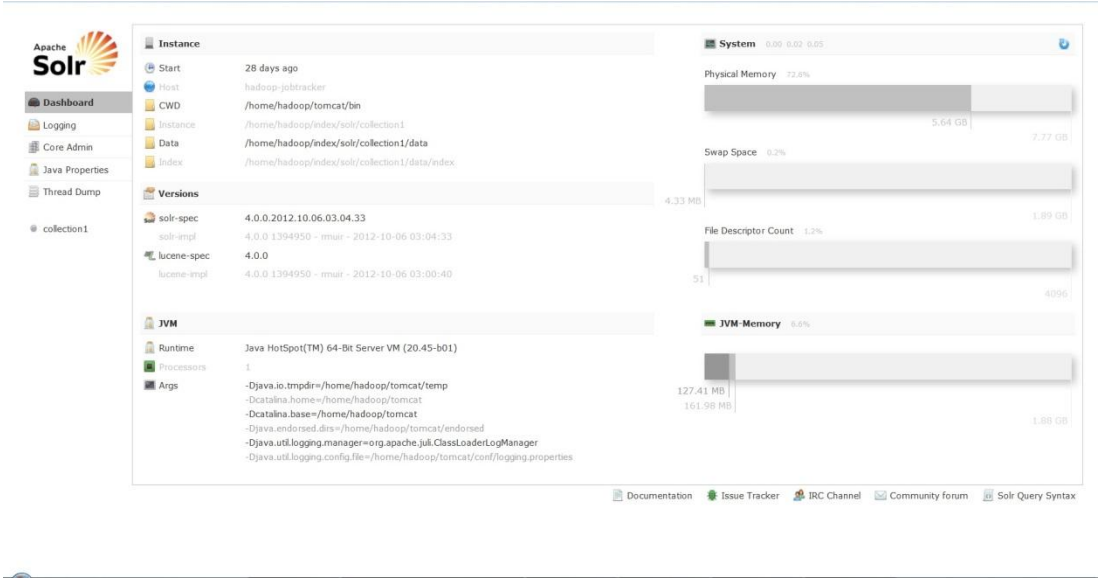
Solr sorgulama işlemlerini gerçekleştirmek için kendine özgü bir sorgulama diline sahiptir. Bu sorgulama diline örnek vermemiz gerekirse:

Çizelge 2.5. Solr Sorgu Örneği

```
subject:"<http://dbpedia.org/ontology/VideoGame>"AND
predicate:"<http://www.w3.org/2000/01/rdf-schema#label>"
```

Yukarıdaki örnekte subjecti “videoGame” olan ve predicate’i “label” olan üçlülerin sonuçları döndürülmektedir. Bu şekilde sorgu ile sadece istenilen üçlülere ulaşılabilmekte ve bu üçlüler üzerinde istenilen işlemler gerçekleştirilebilmektedir.

Ayrıca Solr’ın tarayıcı üzerinde ulaşıp genel performansını gözlemleyebileceğimiz, optimizasyon ayarlarını yapıp çalıştırmak istediğimiz sorguları çalıştırabileceğimiz bir yönetim konsolu da bulunmaktadır. Aşağıdaki şekilde bu yönetim konsolunun ekran görüntüsü bulunmaktadır. Uygulama dışından direk olarak Solr üzerinden işlemleri gerçekleştirmek için bu yönetim konsolu kullanabilmektedir.



Şekil 2.3. Solr Yönetim Ekranı

2.8. HIVE

Hive Hadoop çatısı üzerine konumlandırılmış bir veri ambarı mimarisidir. Amacı SQL dil bilgisine sahip analistlerin Hadoop dosya sistemi üzerindeki verilerde analiz işlemlerini gerçekleştirebilmesidir[13] . Facebook tarafından geliştirilmiştir.

Mapreduce görevi gerçekleştirmek için Java kodlama bilgisi gerekmektedir. Hive ile Java kodlama bilgisine sahip olmayan kişilerinde MapReduce görevi oluşturabilmesi sağlanmış olmaktadır.

Hive Hadoop dosya sistemi üzerindeki dosyaları veritabanı benzeri bir yapı üzerinde tutar. Dosyalar hive tabloları ile eşleştirilip sistemde tutulur ve bu veriler üzerinde analiz gerçekleştirilmek istendiğinde HQL sorgu dili ile sorgular gerçekleştirilir.

Hive de 4 farklı veri modeli vardır.

- Veritabanları
- Tablolar
- Bölümler
- Biriktirme Yerleri (Bucket)

Tablolar ilişkisel veritabanlarındaki tablolara karşılık gelir. Dosya sistemindeki her bir klasör hive tablsuna karşılık gelmektedir. Bölümler, indeks yapısına benzer klasörler altındaki alt klasörlerdir. Performans artışı sağlar. Biriktirme yerleri, veriyi hash fonksiyonuna göre böler ve bu şekilde paralel çalıştırma performans artırmayı amaçlar.

Hive tip olarak integer,string ve boolean basit tiplerini ve bunların alt kümelerini destekler. Ayrıca struct, map ve array karmaşık tiplerini de destekler.

2.9. PIG

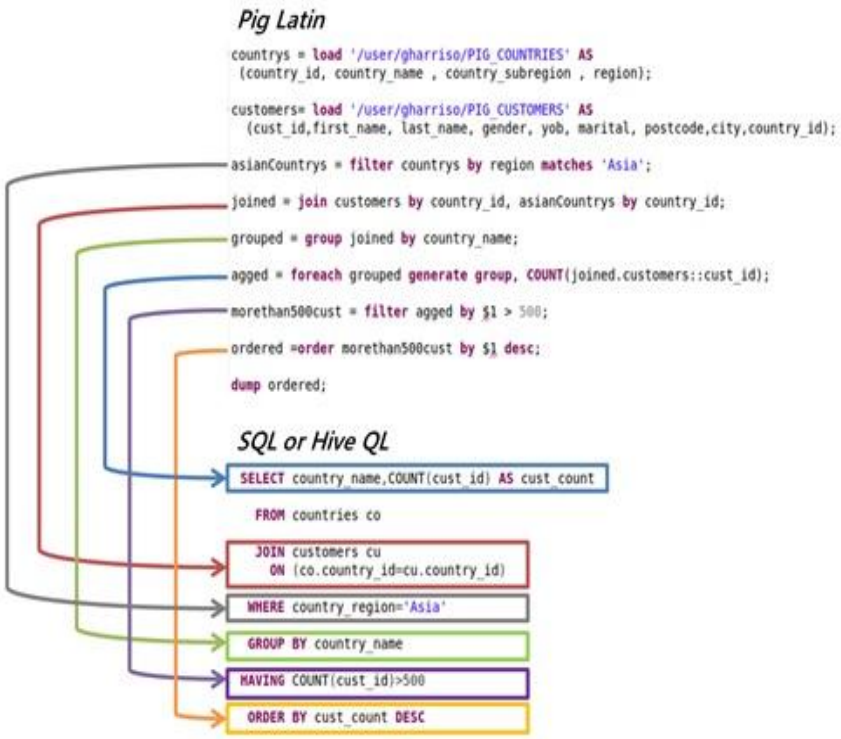
Pig, Yahoo tarafından geliştirilmiş ve MapReduce görevini Java ile kodlamak yerine PigLatin adı verilen bir veri akış diliyle yazılabilmesini sağlayan bir Hadoop parçasıdır. MapReduce kodunu yazabilmek için Java bilgisi gerekmekte ve bazı

görevlerde karmaşık kodların oluşturulabilmesi gerekmektedir. Pig ile daha basit bir veri akış diliyle bu işlemlerin gerçekleştirilebilmesi sağlanmaktadır.

PigLatin veri akış diliyle yazılan kodlar MapReduce görevlerine çevrilip Hadoop üzerinde çalıştırılmaktadır. Pig'e girdi olacak dosyalarda yine HDFS üzerinde saklanmaktadır. Sağladığı kolay kullanılabilirlik, optimizasyon fırsatları ve kullanıcı tanımlı fonksiyonlar ile Hadoop üzerinde paralel süreç çalıştırma da önemli rahatlıklar sağlamaktadır.

Pig'i Hadoop kullanan çok sayıda önemli şirket kullanmaktadır. Başta Twitter, LinkedIn gibi sosyal ağlar analiz için Pig kullanmaktadır. Bu siteler yer alan arkadaş öneri sistemleri Pig ile yapılan analiz sonuçlarında ortaya çıkmaktadır.

PigLatin dilini anlayabilmek için SQL ile karşılaştırmasını aşağıdaki şekilde görebiliriz. Bu örnekte PigLatin'de öncelikle "load" komutu ile değişkenlerin üstüne değerlerin alındığını daha sonra da diğer programlama dillerinden alışık olduğumuz "filter", "join" ,"group" ,"foreach" ve "orderBy" komutlarıyla istediğimiz işlemleri gerçekleştirebildiğimizi görüyoruz.



Şekil 2.4. Pig vs SQL

3. MAPREDUCE İLE RDFS ÇIKARSAMA İŞLEMİNİN GERÇEKLEŞTİRİLMESİ

Daha önceki bölümlerde çıkarsama hakkında teorik bilgiler verdik. RDFS çıkarsama işlemi sırasında kullanılacak kuralları Çizelge halinde tanımladık. Bu bölümde bu çıkarsama işlemlerinin MapReduce[16] ile nasıl gerçekleştirmemiz gerektiğini ve hangi yöntemler ile nasıl performans artışları sağlayabileceğimizden bahsedeceğiz.

3.1. Kuralların Eliminasyonu

RDFS çıkarsama işleminde kullandığımız 14 adet kural bulunmakta bu kurallar Çizelge 2-4 de görülmektedir. Bu kurallardan bazıları önemsiz ve çıkarsama işlemi sırasında göz ardı edebileceğimiz kurallardır [1]. Bu kuralları dikkate almadan çıkarsama yapabilmemizin sebebi, kuralların çıkarsama işleminde kullanıldığında ortaya çıkan sonuçların herhangi bir çıkarsama kuralında girdi olarak kullanılmamasıdır. Yani bu kurallar herhangi bir sıralama ya da çalışma ilkesine gerek kalmadan istenilen bir zamanda çalıştırılabilirler. Örnek olarak diğer kuralların çalışması bittikten sonra basit bir MapReduce görevi çalıştırarak bu kurallar içinde çıkarsama işlemi tamamlanabilir. Çizelgede bu yapıdaki görevlere baktığımızda kural 1, 4a, 4b, 6 ve 10 karşımıza çıkmaktadır. Bu kuralların neden bu gruba girdiğini daha iyi açıklamak için bir örnek gerçekleştirecek olursak:

Çizelge 3.1. RDFS Çıkarsama Kuralı 4

4a	s p o	o rdf:type rdfs:Resource
4b	s p o	s rdf:type rdfs:Resource

Çizelgedeki 4a ve 4b kurallarına bakacak olursak subjecti ya da objecti rdfs:Resource olarak tanımlanan tüm üçlüler bu kural tanımına uymaktadır. Bu kuralın üçlülere uygulanması ve sonuç alınması oldukça kolaydır ancak bu kural sonucunda ortaya çıkan rdfs:Resource tanımlı üçlü herhangi bir çıkarsama işlemi için kullanılmamaktadır. İlk olarak bakıldığında sanki kural 2, 3, 7 ve 9 kurallarına girdi

olabilecek gibi durmaktadır ancak bunlar içinde girdi olamamaktadır. Kural 2 ya da 3 e girdi olabilmesi için rdf:type değerinin domain ya da range ilişkisi tanımı olması gerekmektedir. Aynı şekilde kural 7 ve 9'a girdi olabilmesi içinde rdf:type ya da rdfs:resource değerleri için alt sınıf ilişkisinin tanımlı olması gerekmektedir. Ancak rdf:type ve rdfs:resource değerleri standart RDF dilinde tanımlanmış olan değerlerdir ve bu değerlerin bu şekilde ilişki tanımlarının bulunmadığı bilinmektedir. Eğer girdi olarak sisteme verilen RDF tanımlarında bu şekilde ilişki tanımlanmış olsa bile bunlar dikkate alınmamalıdır çünkü bunun sonucunda "ontoloji hijacking" problemi ortaya çıkabilmektedir. Sonuç olarak bu kuralların çıktısında oluşan üçlülerimizin herhangi bir kurala girdi olamayacağını görmüş oluyoruz. Bu durumdaki diğer kurallarda incelenecek olursa bu şekilde bir durumla karşılaşıldığı görülmektedir. Bu sebepten ötürü çıkarsama işlemi gerçekleştirilirken bu kurallar dikkate alınmadan işlem gerçekleştirilebilir. Bu durumdan sonra da çıkarsama işleminde dikkate almamız gereken 9 adet kuralımız bulunmaktadır [1].

3.2. Kuralların Çalışma Sırası

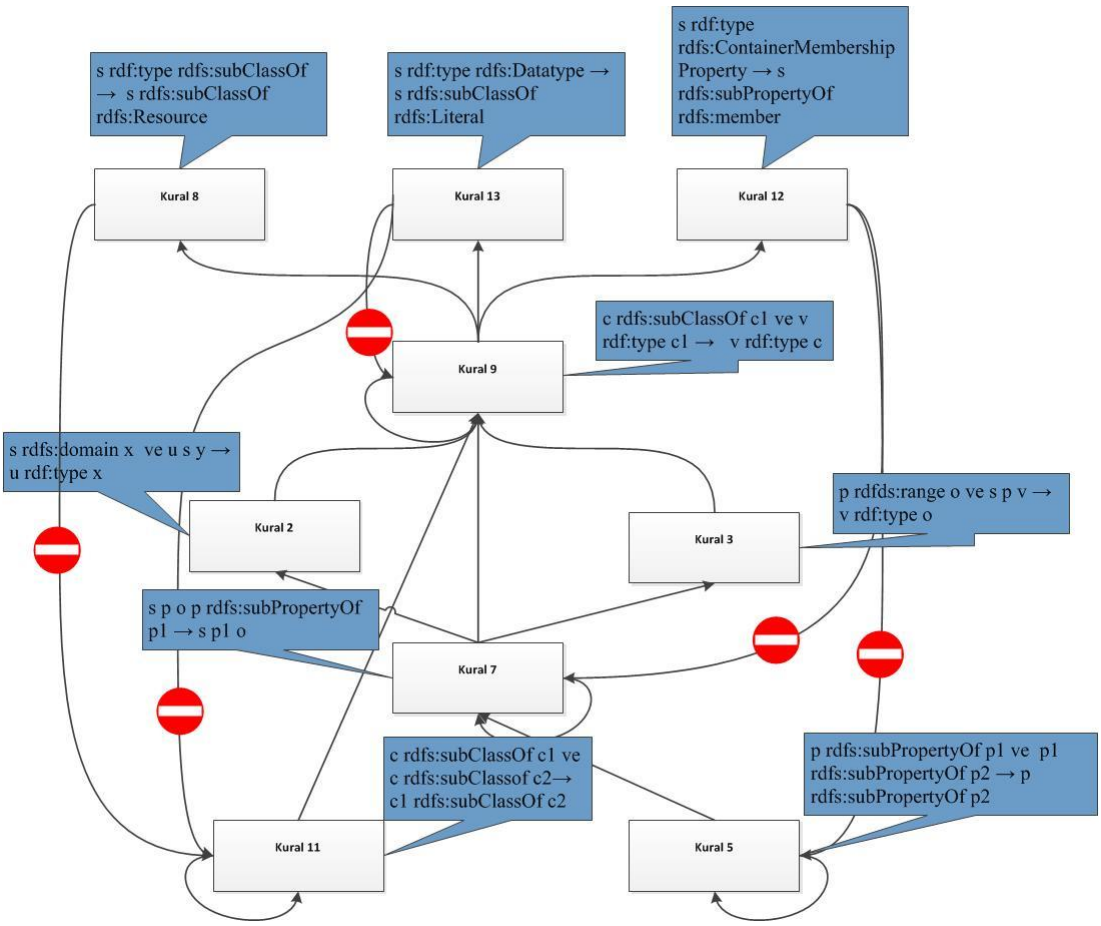
Bazı kuralları çıkarsama işleminde dikkate almayacağımızı yukarıdaki bölümde açıklamıştık. Kalan 9 kural içinse hangi kuralın sonucunda nasıl çıktıların oluştuğu ve hangi kuralların aynı MapReduce görevinde çalışabileceği ve bu MapReduce görevlerinin çalışma sıralamaları belirlenmelidir. Bu 9 kural incelendiğinde 4 farklı çıktı oluştuğu görülmektedir [1]. Kural 5 ve 12 sonucunda alt özellik ilişkisi oluşturan şema üçlüleri oluşmaktadır. Kural 8,11 ve 13 sonucunda alt sınıf ilişkisi olan üçlüler oluşmaktadır. Kural 2,3 ve 9 sonucunda rdf:type tipiyle ilişkisi veri üçlüleri oluşmaktadır. Son olarak kural 7 sonucunda da genel bir yapıya uymayan bir üçlü oluşabilmektedir.

Çıkarsama yapılan kuralların öncül değerlerine göre de incelersek daha kesin bir şekilde kuralları parçalara bölebiliriz. Kural 5 ve 10'u ilk olarak ele alırsak bu kuralların sadece şema üçlülerinde oluştuğunu ve alt özellik ile alt sınıf ilişkisi tanımladığını görüyoruz. Kural 8, 9, 12 ve 13 de ise "predicate" olarak rdf:type, rdfs:subClassOf veya rdfs:subPropertyOf kullanıldığını görüyoruz. Sadece kural 2,3 ve 7 genel olarak tüm üçlüleri kapsamaktadır.

Kuralları bu şekilde sınıflandırmak bize hangi kuralların hangi görevlerde çalıştırılabileceğini ve böylelikle görev sayısını en aza indirgeyerek tam bir çıkarsama işlemini gerçekleştirebileceğimizi göstermektedir. Ayrıca bu görevlerin çalışma sırası da yine bu sınıflandırmaya göre belirlenecektir. Aşağıdaki şekilde bu kurallar arasındaki ilişki gösterilmektedir.

İdeal görev çalışma sırası şeklin en altından başlayıp yukarı doğru devam edecek şekilde olmalıdır. Buna göre görevlerin çalışması sırası öncelikle kural 5 ve 11 çalıştırılmalı daha sonra sırasıyla kural 7, kural 2 ve kural 3 çalıştırılmalı, bu kurallardan sonra kural 9 çalıştırılmalı son olaraksa kural 8, 12 ve 13 çalıştırılmalıdır.

İlk olarak bakıldığında kural 8,12 ve 13 sonucunda oluşan çıktıların kural 11 ve 5' in girdisi olabileceği düşünülmektedir. Ancak dikkatli incelendiğinde böyle bir durumun olamayacağı görülmektedir. Kural 8 sonucu oluşan çıktı kural 11 ve kural 9'a girdi olabilir. Ancak burada kural 9 durumunu yok sayabiliriz çünkü bunun sonucunda oluşan sonuç x in tipi rdfs:resource durumunda olan tüm değerler için geçerlidir. Böyle bir çıkarsama işlemine gerek yoktur. Kural 11 durumunu da dikkate almayabiliriz. Bunun sebebi de rdfs:Resource 'un herhangi bir alt sınıf tanımı bulunmamaktadır. Bu yüzden bu yönden bir çıkarsama işlemi gerçekleştirilememektedir.



Şekil 3.1. Kuralların Çalışma İlişkileri

Kural 12 'in çalıştırılması sonucunda rdfs:subPropertyOf rdfs:member üçlü yapısı oluşmaktadır. Bu çıktı sonucunda kural 5 ya da kural 7 çalıştırılabilir. Ancak iki durumda da çıktı sonucu oluşan üçlü bir daha kullanılamaz. Aynı durum kural 13 içinde geçerlidir. Bunun çıktısı olarak kural 9 ya da kural 11 çalıştırabilir ancak sonucunda s rdfs:subClassOf rdfs:Literal veya s rdfs:type rdfs:Literal üçlüleri oluşmaktadır. Bunun sonucunda da herhangi bir çıkarsama yapılamayacağı için bu kurallarda dikkate alınmamaktadır.

Kuralların çalışma sırasını belirlerken en önemli durum kurallar arasında döngü oluşturulmamasıdır. Yani bir kural için görev çalıştırıldıktan sonra bir daha bu görevin çalıştırılmaması gerekmektedir. Bu durumda son olarak kural 8, 12 ve 13 çalıştırdıktan sonra oluşturduğumuz üçlüler herhangi bir kurala girdi olmamaktadır. Bu durumda gerçekleştirdiğimiz çıkarsama işlemi için çözüm noktasına ulaşmış oluyoruz. Bu çıkarsama işlemi bittikten sonra bir önceki başlıkta tartışıp Çizelgeden

çıkardığımız kuralları istediğimiz sırada çalıştırarak tam olarak RDFS çıkarsama işlemini tamamlamış oluruz.

Son olarak sadece kuralların çıktılarının kendilerine girdi olduğu durumlarda bir döngü oluştuğunu ve bu döngünün de herhangi bir değer oluşmayana kadar sürmesi gerektiğini gözlemliyoruz. Bu problemi çözmek içinde bizde şema üçlülerini indeksleyip bunun üzerinden çıkarsama işlemini yaparak engellemiş oluyoruz.

3.3. Şema Üçlülerinin İndekslenmesi

RDF dilindeki veri setleri incelendiğinde şema üçlülerinin veri üçlülerinde çok daha az sayıda olduğu gözlemlenmektedir ve genel olarak kural çıkarsamalarında şema üçlüleriyle veri üçlüleri arasında birbirine bağlanacak bir ilişki olduğu görülebilmektedir. Bu nedenle şema üçlülerini indeksleyip buna ait verileri indeksten çekip veri üçlülerini de normal okuyarak bunlar arasında eşleme sağlamak çıkarsama işlemini efektif bir hale getirmektedir.

Kural kümemizde olup da indekslersek performans artışı sağlayacağımız ilişkilere bakacak olursak öncelikle “domain” ve “range” ilişkisini görüyoruz. Bunun dışında alt özellik ve alt sınıf ilişkisindeki üçlüleri de indekslememiz gerekiyor. Yani kural olarak bakarsak kural 2, 3, 5, 7, 9 ve 11 deki şema üçlüleri için indeksleme işlemi gerçekleştiriyoruz.

İndeksleme işlemini gerçekleştirmek için Solr [2] arama sunucusunu kullanıyoruz [11][17]. Üçlüleri kendi isteğimize göre indeksleyebilmek için şema yapılandırma dosyasında eklemeler yapıyoruz. Üçlülerin ana yapısı olan “subject”, “predicate” ve “object” sahalarını Solr şemasına ekliyoruz. Böylece ilgili kayıtları bu sahalar altında indeksleyip istenildiğinde hızlı bir şekilde istediğimiz kayıtlara ulaşılabilmeyi sağlamış oluyoruz.

Yapılan işlemi daha iyi anlamak için bir örnek ile açıklayacak olursak:

Yigit rdfs:domain Student

Cetin rdfs:range Person

Student rdfs:subclassof Person

Üçlü setini ele alırsak bu üçlülerin şema üçlüleri olduğu için sistemde indekslenmesi gerekmektedir. Solr üzerinde aşağıdaki Çizelgeye benzer bir şekilde bu veriler indekste saklanmaktadır.

Çizelge 3.2. Subject Predicate Object

Subject	**Predicate**	**Object**
Yigit	Rdfs:domain	Student
Cetin	Rdfs:range	Person
Student	Rdfs:subclassof	Person

Çıkarsama işleminde hangi kural için çıkarsama yapılacak ise o kural ile ilgili şema üçlüleri indeksten sorgu ile vektöre alınır ve bu yapı kullanılarak çıkarsama işlemi gerçekleştirilir. İndeks üzerinden ilgili verilerin nasıl alındığına örnek verecek olursak:

predicate:"<Rdfs: domain>"

Yukarıdaki sorgu solr sunucusu üzerinde çalıştırıldığın bize "predicate" değeri rdfs:domain olan tüm değerleri döndürmektedir. Bu değerler dosya sistemindeki üçlülerle eşleştirilerek gerçekleştirilmek istenen çıkarsama işlemlerin sonuçları üretilmektedir.

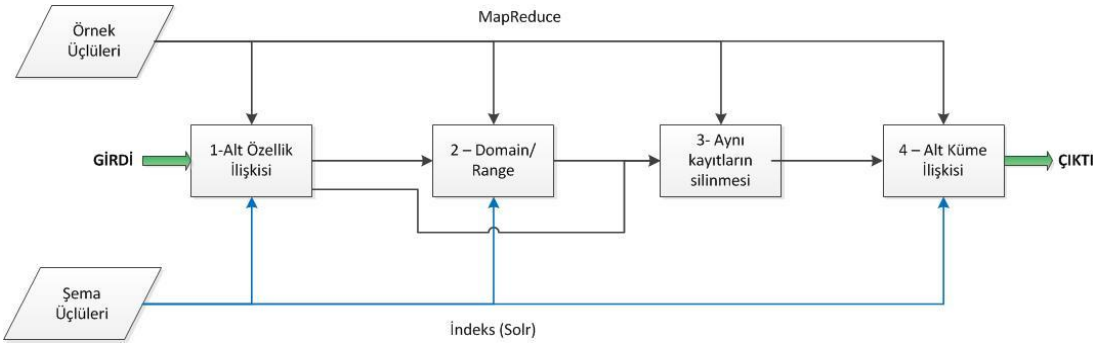
İndekslemenin bize sağladığı avantaj büyük veri kümelerinde ortaya çıkmaktadır. Bir dosyayı tüm olarak hafızaya alıp çıkarsama işlemini gerçekleştirmektense bu dosyayı indekste saklayıp sadece çıkarsama için kullanılacak üçlülere hızlı bir şekilde ulaşmak bir performans artışı sağlamaktadır. Özellikle veri kümelerini tek dosya içinde paylaşan veri kümelerinde (örneğin freebase) şema üçlülerini ayırmak büyük bir yük olmaktadır. Bu verileri hafızaya alma işlemi de yine bir performans kaybına sebep olmaktadır. Bu sebepten bu veri setini bir defa indeksleyip daha sonra indeks verileri üzerinden çıkarsama işlemlerinin gerçekleştirilmesinin performans artışına neden olacağı düşünülmektedir.

Ayrıca yukarıda belirttiğimiz gibi indeksleme kullanılarak verileri vektöre alarak iç içe döngü oluşumunda önüne geçmiş oluyoruz.

3.4. MapReduce Görevleri ve Çalışma Sıraları

Daha önceki bölümlerde kuralların hangi sırayla çalışacağını ve bu kurallar çalıştırılırken ne gibi yöntemler kullanılacağı konularında bahsedildi. Bu bölümde de artık bu yapılar kullanılarak MapReduce görevlerinin gerçekleştirilmesi ve bu görevlerin hangi sırayla çalıştırılması gerektiğinden bahsedilecektir.

MapReduce görevlerinin çalışması sırasına bakarsak öncelikle alt özellik ilişkisi (Kural 5, Kural 7) ile ilgili görev çalıştırılır. Daha sonra "domain" ve "range" ilişkisinin çıkarsamasını yapan görev çalıştırılır. Üçüncü aşama da çift kayıtları temizlemek için bir görev çalıştırılır. Son olarak alt sınıf ilişkisinin çıkarsamasını yapan MapReduce görevi çalıştırılır. Aşağıdaki şekilde bu görev sıralaması görülebilmektedir. Ayrıca aşağıdaki şekilde görevlerin çıktılarının birbirlerine girdi olarak kullanılabildiği görülmektedir.



Şekil 3.2. Görevlerin Çalışma Sırası

3.4.1. Alt Özellik İlişkisi Görevi

Bu görevde kural 5 ve 7 için çıkarsama işlemi gerçekleştirilir. Girdi olan üçlüler teker teker map aşamasına gelecek şekilde düzenlenir. Map aşamasına gelen girdinin predicate değeri "rdf:subpropertyOf " ve object değerinin de alt özellikleri varsa kural 5 için gerçekleştirim yapılacağı anlaşılır. Map aşamasının çıktısı olarak anahtarı değeri bu üçlünün subject değeri olur. Ayrıca bu görevde iki kural

çalıştırıldığı için ve bu kuralları ayırt etmek için işaret değişkeni de anahtar değerine eklenir. Map çıktısının anahtar değerine karşılık gelen değere de üçlünün object değeri atanır.

Map aşaması bittikten sonra reduce aşamasına geçilir. Reduce aşamasında anahtar değerine göre gruplanmış şekilde girdiler alınır. Öncelikle indeks verisinden ilgili şema üçleri çekilir daha sonrada işaret değeri kontrol edilir. İşaret değeri kural 5 için olan işaret değeriyse girdi olan üçlü değerinin subject değerinin alt özellik ilişkisi içinde olduğu özellikler için türetilmiş olan üçlüler oluşturulur.

Kural 7 içinde map aşamasına gelen üçlü girdisinin predicate değerinin alt özellik ilişkisi var ise kural 7 gerçekleştirimi yapılacağı anlaşılır. Map aşamasının çıktısı için anahtar değeri olarak işaret değeri, üçlünün subject ve object değerleri kullanılır. Anahtara karşılık gelen değere de üçlünün predicate değeri atanır. Reduce aşamasında da öncelikle indeks verisinden ilgili şema üçlüleri alınır. Daha sonra da ilk olarak işaret değeri kontrol edilir ve gelen değerin kural 7 için reduce aşamasına yollandığı kontrol edilir. Bu kontrol işlemini başarılı bir şekilde tamamlanırsa üçlünün predicate değeri için süper özellikler hesaplanır ve bu predicate'in süper özellikleri var ise üçlünün subject ve object verilerini bu süper özellikler altında birleştiren yeni üçlüler türetilir.

Sonuç olarak map aşamasında girdiler gruplandırılıp reduce aşamasında da çıkarsama sonucu yeni üçlüler oluşturulmaktadır. Algoritma 1'de de yukarıdaki anlatımı gerçekleştirilen işlemler gözlemlenebilir.

Algoritma 1. Alt Özellik İlişkisi

```
Setup(context) :
 baglanSolrServer()
 subproperties=getSemaUclu("rdfs: subPropertyOf")

Map (key,value) :
if (subproperties.contains(value.predicate)) then  // Kural 7
  key = "1" + value.subject + "-" + value.object
  emit(key, value.predicate)
```

```
if (subproperties.contains(value.object) &&
value.predicate == "rdfs:subPropertyOf") then // Kural 5
  key = "2" + value.subject
  emit(key, value.object)


Reduce(key, iterator values):
switch (key[0])
case 1: // Kural 7 için
   for (predicate in values)
      // subpropertyleri dolaşıp onun süper propertylerini alıyoruz.
       superproperties.add(subproperties.recursive_get(value))
   for (superproperty in superproperties)
       emit(null, triple(key.subject, superproperty, key.object)
case 2: // Kural 5 için
    for (predicate in values)
       / /subpropertyleri dolaşıp onun süper propertylerini alıyoruz.
       superproperties.add(subproperties.recursive_get(value))
    for (superproperty in superproperties)
       emit(null, triple(key.subject, "rdfs:subPropertyOf", superproperty))
```

3.4.2. Domain-Range İlişkisi Görevi

Bu görevde kural 2 ve kural 3 için çıkarsama işlemleri gerçekleştirilmektedir. Bu görev bir önceki görevden farklı olarak iki farklı kuralda aynı küme içinde işlenmektedir. Bunun amacı bu iki görev birbirine çok yakın olduğu için aynı verileri üretmelerin önüne geçmektir. Burada kastedilen bir önceki görevde anahtar değerlerine işaret değerini koyarken bu görevde map çıktı değerine işaret değeri konulmaktadır. Böylece aynı anahtar değerine sahip girdiler aynı reduce fonksiyonunda işlenmektedir.

Girdi olarak gelen üçlü değeri map aşamasında parçalanır ve eğer domain ilişkisi varsa subject değeri, range ilişkisi varsa object değeri map çıktısı için anahtar değeri olarak atanır. İki durum içinde anahtar değerine karşılık gelen değer için predicate ve işaret değeri atanır.

Reduce aşamasında öncelikle ilgili şema değişkenleri indeks yapısından vektöre alınır. Daha sonra da her predicate değeri için domain ve range ilişkisi olup

olmadığına bakılır. Eğer bu predicate değerleri bu ilişkiye sahip ise de reduce aşamasına girdi gelen anahtar değeriyle bu değerler arasında "rdf:type" tanımlı türetilmiş üçlüler oluşturulur. Bu şekilde domain range işlemi için çıkarsama işlemi gerçekleştirilmiş olur. Algoritma 2'de domain range çıkarsama işleminin gerçekleştirimi gözlemlenebilir.

Algoritma 2. Domain Range İlişkisi

```
Setup(context) :
  baglanSolrServer()
  domains=getSemaDomainUclu("rdfs:domain")
  ranges=getSemaDomainUclu("rdfs:range")

Map(key, value):
if (domains.contains(value.predicate)) then // Kural 2
  key = value.subject
  emit(key, value.predicate + "d")
if (ranges.contains(value.predicate)) then // Kural 3
  key = value.object
  emit(key, value.predicate +''r'')

Reduce(key, iterator values):
for (predicate in values)
   switch (predicate.flag)
   case "r": // Kural 3 – predicate değeri için range değerlerini bul
     types.add(ranges.get(predicate))
   case "d": // Kural 2 – predicate değeri için domain değerlerini bul
     types.add(domains.get(predicate))
for (type in types)
 emit(null, triple(key, "rdf:type", type))
```

3.4.3. Aynı Kayıtların Silinmesi

Çalıştırılan iki görevin sonucu olarak ortaya çıkan üçlülerden bazıları girdi ya da çıktı değerinde birden çok defa bulunuyor olabilir. Bu durum gereksiz yer kaybına ve diğer çıkarsama işlemlerinde de performans kaybına neden olacaktır. Bu nedenle bu görevde girdi değerinde bulunup ilk 2 görevin çıktısı sonucu oluşan ya da direk olarak çıktı da birden çok defa tekrarlayan üçlüler sistemden silinir. Bu silinme işleminin algoritması aşağıdaki algoritma-3 de gösterilmektedir.

Algoritma 3. Aynı Kayıtların Silinmesi

```
Map(key, value):
 // Burada value üçlünün değeridir. Aynı üçlüden birden çok varsa aynı reduce
 // fonksiyonuna gider
 emit(value,key)

Reduce(key, iterator values):
 // Birden çok değer gelse bile çıktı da bir tek üçlü oluşturulur.
 emit(null, key)
```

3.4.4. Alt Sınıf İlişkisi Görevi

Bu görevde kural 8, 9, 11, 12 ve 13 çalıştırılacaktır. Burada map aşamasında predicate değeri rdf:type ya da rdf:subclassOf olan üçlülerin subject değerleri anahtar değer olarak object değeri de bu anahtar değere karşılık gelen çıktı değeri olarak atanır ve bu değerler reduce aşamasına taşınır.

Reduce aşamasında da diğer görevlerde olduğu gibi ilgili şema üçlüleri indeks verisinden çekilir. Burada indeks verisinden iki tip şema üçlüsü çekilmektedir. Bunlar rdfs:subclassOf ve rdfs:member predicate ile bağlanmış şema üçlüleridir. İndeks verisinden alınan şema üçlüleri aşağıdaki Algoritma-4 de tanımlandığı gibi kullanılarak çıkarsama işlemi gerçekleştirilir.

Algoritma 4. Alt Sınıf İlişkisi

```
Setup(context) :
  baglanSolrServer()
  subclasses=getSemaUclu(“rdfs:subclassOf”)

Map(key, value):
 if (value.predicate = "rdf:type") then
    key = "0" + value.predicate
    emit(key, value.object)
if (value.predicate = "rdfs:subClassOf") then
    key = "1" + value.predicate
    emit(key, value.object)
```

```
Reduce(key, iterator values):
  for (class in values)
    superclasses.add(subclasses.get_recursively(class))
    switch (key[0])
     case 0: // rdf:Type için çıkarsama
       for (class in superclasses)
         if !values.contains(class)
           emit(null, triple(key.subject, "rdf:type", class))
     case 1: // rdfs:subClassOf için çıkarsama
       for (class in superclasses)
         if !values.contains(class)
           emit(null, triple(key.subject, "rdfs:subClassOf", class)
```

4. HIVE İLE RDFS ÇIKARSAMA İŞLEMİNİN GERÇEKLEŞTİRİLMESİ

Daha önceki bölümlerde Hive'in Hadoop mimarisi üzerine konumlanmış ve sorgu dili ile analiz işlemlerini gerçekleştirebildiğimiz bir yapı olduğunda bahsetmiştik. Bu bölümde RDFS çıkarsama işlemlerini hive ile nasıl gerçekleştirebileceğimiz anlatılacaktır. Daha sonra da Hive'ın performansı değerlendirilecektir.

Öncelikle çıkarsama işleminde kullanılacak girdilerin Hive üzerine tablolar şeklinde tanımlanması gerekmektedir. Burada aynı kayıtların üretiminin önüne geçmek için dosya sistemi çözümündeki yönteme benzer bir yöntem kullanılacaktır. Yani şema üçlüleri ayrı bir tabloda tutulacak diğer üçlüler ayrı bir tablo da tutulacaktır. Daha sonra sorgu yazılırken bunlar arasında birleştirme oluşturularak sonuç elde edilecektir.

Bu tabloların formatı üçlüleri saklayacak şekilde oluşturulmalıdır. Buna göre "subject", "object" ve "predicate" değerleri tablonun sütunlarını oluşturmaktadır. Aşağıdaki sorgu "rdfsInput" adında bir tablo oluşturmakta ve "/user/dataset/" klasörü altındaki dosyalardaki verileri bu tablo içine doldurmaktadır.

Çizelge 4.1. Hive Tablo Oluşturma Sorgusu

```
CREATE TABLE rdfsInput (subject STRING, predicate STRING, object STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
LOCATION '/user/dataset/';
```

Şema üçlüleri içinde aynı şekilde bir Hive tablosu oluşturulur ve ilgili girdiler aşağıdaki sorgu kullanılarak tabloya doldurulur.

Çizelge 4.2. Hive Tablo Oluşturma Sorgusu

```
CREATE TABLE rdfsSema (subject STRING, predicate STRING, object STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
LOCATION '/user/dataset/';
```

Hive tablolarına ilgili kayıtlar alındıktan sonra artık sorgular yazılarak çıkarsama işlemleri gerçekleştirilebilir. Kuralların önceki bölümlerde çalışma sıralarından bahsetmiştik bu çalışma sırasına göre sorguları tanımlarsak öncelikle alt özellik ilişkisi kurallarının sorgularını çalıştırmalıyız. Bunlar kural 7 ve kural 5'dir. Kural 5 deki çıkarsama işleminin sorgusu aşağıdaki gibidir. Burada şema üçlülerinin hepsi rdfsSema tablosundaydı bu yüzden bu tablo kendi içinde birleştirilir. Eğer bu üçlülerin predicate değerleri "subpropertyOf" ise ve birinci üçlünün subject değeri ile ikinci üçlünün object değeri aynı ise buradan yeni bir "subpropertyOf" ilişkisi tanımlanabilmektedir.

Çizelge 4.3. Kural 5 için HQL

```
SELECT CONCAT (rd1.subject, ' ',' <http://www.w3.org/2000/01/rdf-schema#subPropertyOf>',' ',rd2.object)
FROM rdfsSema rd1
LEFT OUTER JOIN rdfsSema rd2 ON (rd1.object=rd2.subject)
WHERE rd1.predicate='<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>'
AND rd2.predicate='<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>'
```

Kural 7 çıkarsama işlemini gerçekleştiren HQL sorgusu da aşağıda belirtilmektedir. Burada şema üçlüleri ile örnek üçlüler arasında birleştirme işlemi gerçekleştirilir. Örnek üçlünün predicate değeri için bir "subPropertyOf" ilişkisi tanımlanmış ise bu ilişkili tanım kullanılarak yeni bir üçlü oluşturabilir.

Çizelge 4.4. Kural 7 için HQL

```
SELECT CONCAT (rd1.subject, ' ',rd2.object,' ',rd1.object)
FROM rdfsInput rd1
LEFT OUTER JOIN rdfsSema rd2 ON (rd1.predicate=rd2.subject)
WHERE rd2.predicate='<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>'
```

Alt özellik ile ilgili çıkarsama işlemleri gerçekleştirildikten sonra sıra "Domain/Range " kurallarının çıkarsama işlemlerini gerçekleştirmeye gelmektedir. Burada Kural 2 ve Kural 3 için HQL sorgularıyla çıkarsama işlemi gerçekleştirilecektir. Kural 2 için HQL sorgusu aşağıda belirtilmektedir. Burada şema üçlüleri ile örnek üçlüleri arasında birleştirme işlemi yapılmaktadır. Predicate değeri "domain" olan şema üçlüsünün, "subject" değeri örnek üçlünün "predicate" değerine eşit ise çıkarsama işlemi gerçekleştirilip yeni bir üçlü oluşturulmaktadır.

Çizelge 4.5. Kural 2 için HQL

```
SELECT CONCAT (rd1.subject, ' ','<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',' ',rd2.object)
FROM rdfsInput rd1
LEFT OUTER JOIN rdfsSema rd2 ON (rd2.subject=rd1.predicate)
WHERE rd2.predicate='<http://www.w3.org/2000/01/rdf-schema#domain>'
```

Kural 3 içinde HQL sorgusu aşağıdaki gibidir. Burada da şema üçlüleri ile örnek üçlüleri arasında birleştirme yapılmıştır. Kural 2 den farkı "domain" ilişkisi yerine "range" ilişkisi için çıkarsama işleminin yapılması ve yeni üçlülerin buna göre oluşturulmasıdır.

Çizelge 4.6. Kural 3 için HQL

```
SELECT CONCAT (rd1.object, ' ','<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',' ',rd2.object)
FROM rdfsInput rd1
LEFT OUTER JOIN rdfsSema rd2 ON (rd2.subject=rd1.predicate)
WHERE rd2.predicate='<http://www.w3.org/2000/01/rdf-schema#range>'
```

Son olarak alt sınıf ilişkisi tanımlayan kurallar için çıkarsama işlemi yapılmalıdır. Bunlarda kural 9, 11, 12 ve 13 olmaktadır. Kural 9 için HQL sorgusu aşağıda belirtilmiştir. Burada şema üçlüleri ve örnek üçlüleri arasında birleştirme işlemi uygulanmaktadır. Predicate değerleri "type" ve "subClassOf" olan üçlüler arasında çıkarsama işlemi gerçekleştirilmektedir. Sonuç olarak predicate değeri "type" olan yeni şema üçlüleri oluşturulmaktadır.

Çizelge 4.7. Kural 9 için HQL

```
SELECT CONCAT (rd1.subject, ' ','<http://www.w3.org/1999/02/22-rdf-syntax-
ns#type>',' ',rd2.object)
FROM rdfsInput rd1
LEFT OUTER JOIN rdfsSema rd2 ON (rd1.object=rd2.subject)
WHERE rd1.predicate='<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
AND rd2.predicate='<http://www.w3.org/2000/01/rdf-schema#subClassOf>'
```

Kural 11 için HQL sorgusu da aşağıda gösterilmektedir. Burada da predicate değeri "subclassOf" olan şema üçlüleri arasında birleştirme işlemi yapılmaktadır. Birinci üçlünün object değeri ile ikinci üçlünün subject değeri birbirine eşit ise "subclassOf" ilişkisini kullanarak çıkarsama işlemi gerçekleştirilmekte ve yeni üçlüler oluşturulmaktadır.

Çizelge 4.8. Kural 11 için HQL

```
SELECT CONCAT (rd1.subject, ' ',rd1.predicate,' ',rd2.object)
FROM rdfsSema rd1
LEFT OUTER JOIN rdfsInput rd2 ON (rd1.object=rd2.subject)
WHERE rd1.predicate='<http://www.w3.org/2000/01/rdf-schema#subClassOf>'
AND rd2.predicate='<http://www.w3.org/2000/01/rdf-schema#subClassOf>'
```

Kural 12 için HQL sorgusu aşağıda belirtilmektedir. Burada herhangi bir birleştirme işlemi yapılmamaktadır. Örnek üçlüleri içinde predicate değeri "type" , object değeri "containerMembershipProperty" olan üçlüler alınmakta ve bu üçlünün subjecti kullanılarak yeni bir üçlü oluşturulmaktadır. Bu üçlü de "member" ilişkisinin bir alt özelliğini tanımlamaktadır.

Çizelge 4.9. Kural 12 için HQL

```
SELECT CONCAT (rd1.subject, ' ','<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>', ' ' ,'<http://www.w3.org/2000/01/rdf-schema#member>')
FROM rdfsInput rd1
WHERE rd1.predicate='<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
AND rd1.object='<http://www.w3.org/2000/01/rdf-schema#ContainerMembershipProperty>'
```

Son olarak Kural 13 için HQL sorgusu aşağıda görülmektedir. Bu sorguda da herhangi bir birleştirme işlemi yapılmamaktadır. Örnek üçlülerinde predicate değeri "type", object değeri "dataType" olan üçlü için çıkarsama işlemi yapılmaktadır. Sonuç olarakta üçlü değerinin subject değeri için "literal" tanımı ile alt sınıf ilişkisi tanımlanmakta ve çıkarsama işlemi gerçekleştirilmiş olmaktadır.

Çizelge 4.10. Kural 13 için HQL

```
SELECT CONCAT (rd1.subject, ' ','<http://www.w3.org/2000/01/rdf-schema#subClassOf>',' '
,'<http://www.w3.org/2000/01/rdf-schema#Literal>')
FROM rdfsInput rd1
WHERE rd1.predicate='<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'
AND
rd1.object='<http://www.w3.org/2000/01/rdf-schema#Datatype>
```

Yukarıdaki sorgular sırasıyla çalıştırıldığında RDFS çıkarsama işlemi başarılı bir şekilde tamamlanmış olur. Ancak yukarıdaki HQL sorguları çalıştırıldığında bunların sonuçlarının da diğer kurallara girdi olabileceği durumlar bulunmaktadır. Bu yüzden HQL sorguları sonucunda oluşan üçlülerin dosya sistemine yazılması gerekmektedir. Bunun için sorgulara sonucun diske yazılmasını sağlayacak olan HQL sorgusu eklenmesi gerekmektedir. Aşağıdaki kod parçası ile bu işlem gerçekleştirilmektedir.

Çizelge 4.11. Dosya Yazma HQL

```
INSERT OVERWRITE DIRECTORY '/user/sampleOutput/container'
```

Hive üzerinde yapılan bu çıkarsama işleminin performansını bazı ayarlamalar ile artırmak mümkündür. Daha önceki bölümde bahsedilen “bucket” yapısı ile paralel çalıştırma da performans artışı sağlanabilmektedir. Oluşturulan tabloların aynıları “bucket” yapısı ile oluşturulup aynı sorgular çalıştırıldığında performansta %50’ye varan performans artışı sağlanabilmektedir.

5. PIG İLE RDFS ÇIKARSAMA İŞLEMİNİN GERÇEKLEŞTİRİLMESİ

Daha önceki bölümlerde Pig'den ve nasıl kullanılabileceğinden bahsetmiştik. Bu bölümde Pig ile RDFS verilerinin nasıl işlenebileceğini[15] ve RDFS çıkarsama işlemlerini, Pig ile nasıl gerçekleştirebileceğimiz anlatılacaktır.

Öncelikle çıkarsama işleminde kullanılacak girdilerin Pig değişkenleri üzerine alınması gerekmektedir. Yapılmak istenen işlemler bu değişkenler üzerinde yapılacaktır. Burada aynı kayıtların üretiminin önüne geçmek için dosya sistemi çözümündeki yönteme benzer bir yöntem kullanılacaktır. Yani şema üçlüleri ayrı bir değişkende tutulacak diğer üçlüler ayrı bir değişkende da tutulacaktır. Daha sonra PigLatin kodu yazılırken bu değişkenler üzerinde çıkarsama işlemini gerçekleştirebileceğimiz gerekli yapılar oluşturulacaktır.

Pig ile her kural için bir kod yazılması gerekmektedir. Burada şekil 3-1 de belirtilen kural çalışma sırası kullanılacaktır. Yani öncelikle alt özellik ilişkisi üzerinden çıkarsama işlemlerini gerçekleştiren kural 5 ve kural 7 için çıkarsama işleminin gerçekleştirilmesi gerekmektedir.

Kural 5 için çıkarsama işlemini gerçekleştiren PigLatin kodu aşağıda verilmiştir. Burada öncelikle değişkenler ile gerekli girdiler eşlenmektedir. Daha sonra sadece alt özellik ilişkisi tanımlayan şema üçlüleri süzülür. Bu adımdan sonra da şema üçlüleri arasında birincisinin object değeri ile diğerinin subject değeri arasında aynı olanlar alınır ve değişkene atanır. Bu değişken döngü ile dolanılır ve RDFS çıkarsama sonucunda oluşan üçlü değerleri oluşturulur. Son olarak sonuç değerleri dağıtık dosya sistemi üzerinde saklanır.

Çizelge 5.1. Kural 5 Pig Kodu

```
rdfsSema = LOAD '/user/rdfsSema/*.*' using PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsSema2 = LOAD '/user/rdfsSema/*.*' using PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsSemaFilter = FILTER rdfsSema BY predicate ==
'<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>';
rdfsSemaFilter2 = FILTER rdfsSema2 BY predicate ==
'<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>';
joined = JOIN rdfsSemaFilter BY object, rdfsSemaFilter2 BY subject;
result = FOREACH joined GENERATE CONCAT(rdfsSemaFilter::subject,'
<http://www.w3.org/2000/01/rdf-
schema#subPropertyOf>'),rdfsSemaFilter2::object;
STORE result INTO '/user/resultPig/subproperty';
```

Kural 7 için çıkarsama işlemini gerçekleştiren PigLatin kodu aşağıda görülmektedir. Burada da öncelikle gerekli girdiler dosya sistemi üzerinden değişkenler üzerine alınmaktadır. Çift kayıtları önlemek için şema üçlüleri ve örnek üçlüler ayrı değişkenlere atanır. Şema değişkenleri içinden sadece alt özellik ilişkisi tanımlayanlar süzülür. Daha sonra da bunlar örnek üçlülerin predicate değeriyle şema üçlülerinin subject değeri eşlenecek şekilde düzenlenir ve bir değişkene atanır. Bu adımdan sonra bu değişkendeki veriler tek tek gezilerek çıkarsama sonucu oluşturulup dağıtık dosya sisteminde belirtilen dosya yolunda saklanır.

Çizelge 5.2. Kural 7 Pig Kodu

```
rdfsSema = LOAD '/user/rdfsSema/*.*'  USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsType = LOAD '/user/rdfsInput/*.*' USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsSemaFilter = FILTER rdfsSema BY predicate ==
'<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>';
joined = JOIN rdfsSemaFilter BY subject, rdfsType BY predicate;
result = FOREACH joined GENERATE
CONCAT(rdfsType::subject,rdfsSemaFilter::predicate),rdfsType::object;
STORE result INTO '/user/resultPig/subproperty2';
```

Alt özellik ile ilgili çıkarsama işlemleri gerçekleştirildikten sonra sıra "Domain/Range " kurallarının çıkarsama işlemlerini gerçekleştirmeye gelmektedir.

Bu ilişki tanımları kural 2 ve kural 3 de görülmektedir. Kural 2 için PigLatin kodu aşağıda görülmektedir. Burada öncelikle gerekli girdiler değişkenler üzerine alınmakta daha sonra şema üçlülerinden sadece domain ilişkisinde olanlar süzülmektedir. Süzülen şema üçlülerinde subject değeri örnek üçlülerin predicate değerinde geçen değerler alınıp değişkene atanmaktadır. Sonrada bu değişkendeki değerler tek tek dolaşılarak RDFS çıkarsama sonucunda oluşan üçlüler oluşturulmakta ve dağıtık dosya sisteminde saklanmaktadır.

Çizelge 5.3. Kural 2 Pig Kodu

```
rdfsSema = LOAD '/user/rdfsSema/*.*' USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsType = LOAD '/user/rdfsInput/*.*' USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsSemaFilter = FILTER rdfsSema BY predicate ==
'<http://www.w3.org/2000/01/rdf-schema#domain>';
joined = JOIN rdfsSemaFilter BY subject, rdfsType BY predicate;
result = FOREACH joined GENERATE CONCAT(rdfsType::subject,'
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'),rdfsSemaFilter::object;
STORE result INTO '/user/resultPig/domain';
```

Kural 3 için PigLatin kodu aşağıda belirtilmektedir. Öncelikle dosya sistemi üzerindeki girdiler belirlenen değişkenler üzerine alınır. Daha sonra şema üçlüleri içinden range ilişkisinde olanlar süzülür. Süzülen şema üçlülerin subject değeriyle örnek üçlülerin predicate değeri aynı olan üçlüler eşlenir ve değişkene atanır. Bu değişkendeki değerler tek tek dolaşılarak Kural 3 için RDFS çıkarsama işleminin sonucu oluşturulur ve dağıtık dosya sisteminde ilgili dosya yolunda saklanır.

Çizelge 5.4. Kural 3 Pig Kodu

```
rdfsSema = LOAD '/user/rdfsSema/*.*' USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsType = LOAD '/user/rdfsInput/*.*' USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsSemaFilter = FILTER rdfsSema BY predicate ==
'<http://www.w3.org/2000/01/rdf-schema#range>';
joined = JOIN rdfsSemaFilter BY subject, rdfsType BY predicate;
result = FOREACH joined GENERATE CONCAT(rdfsType::object,'
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'),rdfsSemaFilter::object;
STORE result INTO '/user/resultPig/range';
```

Son olarak alt sınıf ilişkisi tanımlayan kurallar için çıkarsama işlemi yapılmalıdır. Bunlarda kural 9, 11, 12 ve 13 olmaktadır. Kural 9 için PigLatin kodu aşağıda belirtilmiştir. Daha önceki kural gerçekleştirimde olduğu gibi öncelikle gerekli girdiler değişkenler üzerinden alınır. Daha sonra şema üçlüleri içinden alt sınıf ilişkisine sahip olanlar süzülür. Süzülen bu şema üçlülerinden subject değeri örnek üçlülerin object değerine eşit olanlar eşlenir ve değişkene atanır. Bu değişken döngü ile dönülerek RDFS çıkarsama sonucu oluşturulur ve dosya sisteminde saklanır.

Çizelge 5.5. Kural 9 Pig Kodu

```
rdfsSema = LOAD '/user/rdfsSema/*.*' USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsType = LOAD '/user/rdfsInput/*.*' USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsTypeFilter = FILTER rdfsType BY predicate ==
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>';
rdfsSemaFilter = FILTER rdfsSema BY predicate ==
'<http://www.w3.org/2000/01/rdf-schema#subClassOf>';
joined = JOIN rdfsTypeFilter BY object, rdfsSemaFilter BY subject;
result = FOREACH joined GENERATE CONCAT(rdfsTypeFilter::subject,'
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'),rdfsSemaFilter::object;
STORE result INTO '/user/resultPig/subclassType';
```

Kural 11 için PigLatin kodu aşağıda görülmektedir. Burada diğer PigLatin gerçekleştirimlerinde olduğu gibi gerekli şema üçlüleri değişken üzerine

alınmaktadır. Daha sonra bu şema üçlülerinden sadece alt küme ilişkisi tanımlayanlar süzülmekte ve bu süzülen değerlerin object değerleriyle subject değerleri aynı olanlar eşleştirilmektedir. Eşleşen bu değerler bir değişkene kayıt edilmekte ve bu değişkendeki değerler tek tek ele alınarak çıkarsama işlemi sonucunda oluşan üçlüler oluşturulmaktadır.

Çizelge 5.6. Kural 11 Pig Kodu

```
rdfsSema = LOAD '/user/yedekSema/*.*' USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsSema2 = LOAD '/user/yedekSema/*.*' USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsSemaFilter2 = FILTER rdfsSema2 BY
predicate=='<http://www.w3.org/2000/01/rdf-schema#subClassOf>';
rdfsSemaFilter = FILTER rdfsSema BY predicate ==
'<http://www.w3.org/2000/01/rdf-schema#subClassOf>';
joined = JOIN rdfsSemaFilter2 BY object, rdfsSemaFilter BY subject;
result = FOREACH joined GENERATE CONCAT(rdfsSemaFilter2::subject,'
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>'),rdfsSemaFilter::object;
STORE result INTO '/user/resultPig/subclass';
```

Kural 12 için PigLatin kodu aşağıda görülmektedir. Burada sadece örnek üçlüleri arasında "ContainerMembershipProperty" tipinde olan kaynaklar için bir çıkarsama işlemi gerçekleştirilmektedir. Bu durumdaki üçlüler süzülüp bir değişkene atılmakta daha sonra bu değişkendeki değerler için "member" ile alt özellik ilişkisi oluşturacak şekilde çıkarsama sonuçları oluşturulmaktadır. Son olarak oluşan bu üçlüler dağıtık dosya sisteminde saklanmaktadır.

Çizelge 5.7. Kural 12 Pig Kodu

```
rdfsType = LOAD '/user/rdfsInput/*.*' USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsTypeFilter = FILTER rdfsType BY predicate ==
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>' AND
object=='<http://www.w3.org/2000/01/rdf-
schema#ContainerMembershipProperty>';
result = FOREACH rdfsTypeFilter GENERATE CONCAT(subject,'
<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>
<http://www.w3.org/2000/01/rdf-schema#member>');
STORE result INTO '/user/resultPig/container';
```

Kural 13 için çıkarsama işlemini gerçekleştiren Pig kodu aşağıda belirtilmektedir. Bu kuralda da sadece örnek üçlüleri içinden çıkarsama işlemi yapılmaktadır. “DataType” tipinde olan kaynaklar için çıkarsama işlemi gerçekleştirilmekte ve bu değer için “Literal” ile alt sınıf ilişkisi tanımlanmaktadır.

Çizelge 5.8. Kural 13 Pig Kodu

```
rdfsType = LOAD '/user/rdfsInput/*.*' USING PigStorage(' ')
AS (subject:chararray, predicate:chararray, object:chararray);
rdfsTypeFilter = FILTER rdfsType BY predicate ==
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>' AND
object=='<http://www.w3.org/2000/01/rdf-schema#Datatype>';
result = FOREACH rdfsTypeFilter GENERATE CONCAT(subject,'
<http://www.w3.org/2000/01/rdf-schema#subClassOf>
<http://www.w3.org/2000/01/rdf-schema#Literal>');
STORE result INTO '/user/resultPig/datatype';
```

6. PERFORMANS DEĞERLENDİRME

Bu bölümde önceki bölümlerde bahsettiğimiz çözümlerin performanslarını gözlemleyeceğiz ve hangi durumlarda hangi çözümün en efektif şekilde olduğunu belirteceğiz.

6.1. Deneylerin Gerçekleştirildiği Test Ortamları

RDFS çıkarsama işlemini gerçekleştirdiğimiz çözümlerin hepsi Hadoop kümesi üzerinde çalışmaktadır. Bu durumda Hadoop kümesinin kaç düğümden oluştuğu ve bu düğümlerin her birinin özellikleri ve aralarındaki ağ hızları performans ölçümlerini etkilemektedir. Aynı zamanda bu düğümler sanal sunucu üzerinde ise sanal sunucunun kendi durumu da çıkarsama işleminin hızını etkilemektedir.

Bu tez kapsamında test işlemlerini gerçekleştirirken öncelikle tek bir düğüm üzerinde Hadoop sistemi kurularak işlemler bunun üzerinde gerçekleştirilmiştir. Bu makine çift çekirdekli bir işlemci, 4 GB RAM ve 500 GB sabit diskten oluşmaktadır. Performans kısmından tek bir düğüm üzerinde çalıştırıldığı denildiğinde bu özellikteki makinada çalıştırıldığı anlamına gelmektedir.

İkinci durumda Hadoop kümesi sanal sunucu üzerinde oluşturulmuştur. Sanal sunucudaki her bir düğüm çift çekirdekli işlemci, 8 GB RAM ve 250 GB sabit diskten oluşmaktadır. Sanal sunucunun hepsi RAID 10 ile yapılandırılıp sanal sunucudaki her bir makine de bu disk yapısını kullanmaktadır.

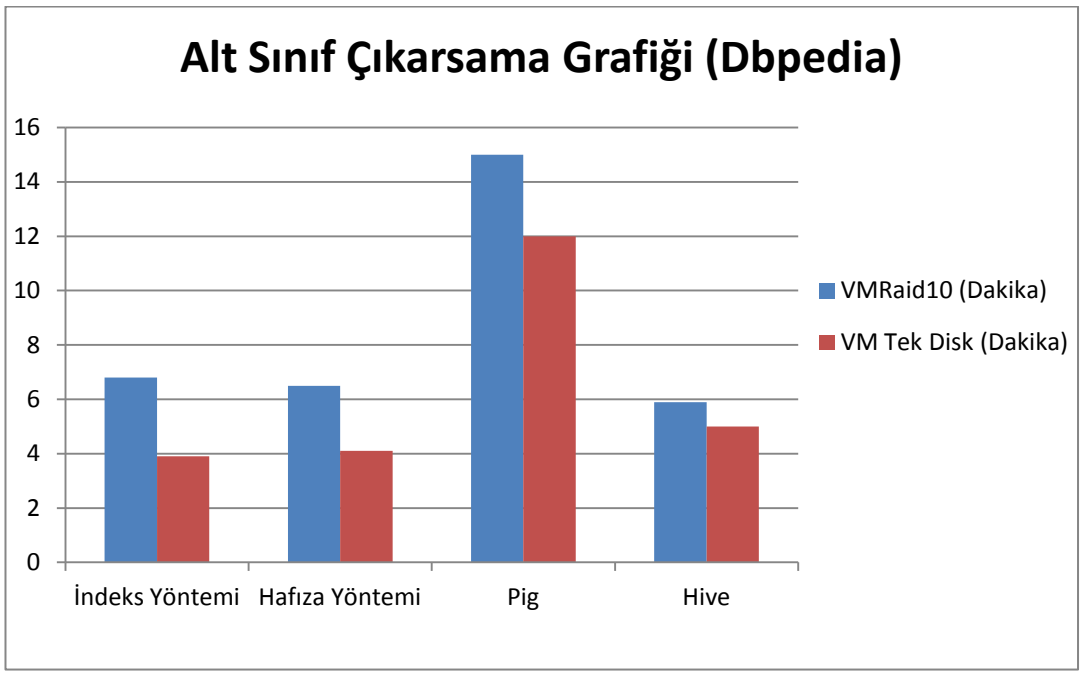
Üçüncü durumda da Hadoop kümesi yine sanal sunucu üzerinde oluşturulmuştur. Ancak bu sefer her bir düğüm için fiziksel olarak bir adet disk ayrılmıştır. Böylece sanal sunucu da karşılaşılabilecek disk IO problemlerinin önüne geçilmesi amaçlanmıştır.

6.2. Performans Sonuçları

Bu bölümde farklı Hadoop kümeleri üzerinde RDFS çıkarsama işlemine yapılan farklı yaklaşımların nasıl sonuçlar verdiği belirtilecektir. Performans sonuçları her bir çıkarsama kuralına göre ayrı ayrı incelenecektir. Bu test sonuçları Dbpedia ve Freebase veri kümeleri üzerinde çalıştırılmıştır. Dbpedia veri kümesinde yaklaşık 85 milyon üçlü kullanılmıştır. Freebase'de iki veri kümesi üzerinde çıkarsama işlemleri çalıştırılmıştır. Birinci durumda tüm veri kümesi üzerinde çıkarsama işlemleri gerçekleştirilmiştir. Bu yaklaşık 2 bilyon üçlüden oluşmaktadır. Ancak bu kadar büyük veri tek bir dosya olarak verildiği için hafızaya alma, Pig ve Hive yöntemleri kullanılamamaktadır. Çünkü bu yöntemlerde şema üçlülerinin ayrı bir şekilde bulunması gerekmektedir. Bu nedenle bu veri kümesi için sadece indeks yöntemi ile çıkarsama gerçekleştirilmiştir. İkinci durumda Freebase veri kümesinin içinden yaklaşık 33 milyon üçlü rastgele seçilmiştir ve bu veri kümesi üzerinden yukarı da anlatılan dört yöntem ile çıkarsama işlemi gerçekleştirilmiştir.

6.2.1. Alt Sınıf İlişkisi İçin Performans Sonuçları

Alt sınıf ilişkisinin de çıkarsama işlemini gerçekleştirmek için toplam 4 kural olduğundan daha önceki bölümlerde bahsedilmiştir. VM'deki Hadoop üzerinde bu kurallar için Dbpedia veri kümesinde çıkarsama işlemi gerçekleştirildiğinde aşağıdaki grafikteki gibi sonuçlar elde edilmektedir. Bu çıkarsama işleminin sonucunda yaklaşık 8,4 milyon üçlü oluşmaktadır. Hive ve Pig sonucunda oluşan çıktı sayısı indeks ve hafıza yöntemine göre yaklaşık 50 bin kayıt az olmaktadır. Bunun sebebi içe içe çıkarsama işlemi yapılması gereken Kural – 5'dir. Bu kural sonucunda üretilen üçlüler aynı kurala girdi olabilmektedir.

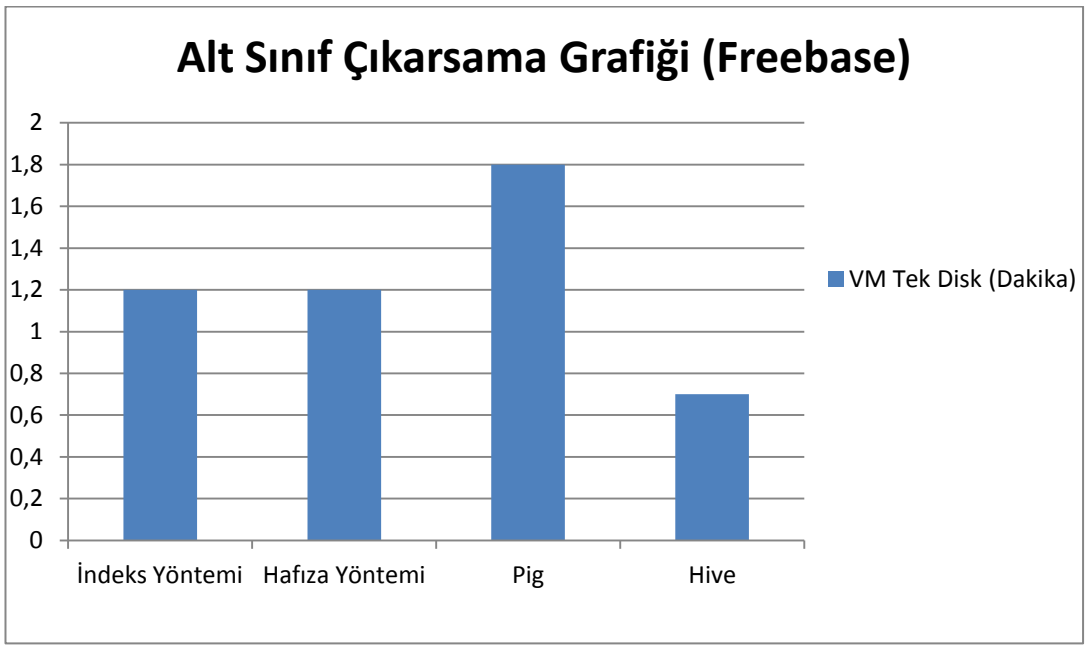


Şekil 6.1. Alt Sınıf Çıkarsama Grafiği (Dbpedia)

Burada görüldüğü gibi çıkarsama işlemini RAID10 durumunda en hızlı şekilde Hive ile gerçekleştirilmektedir. Bunun yanından bizim önerdiğimiz indeksleme yöntemi, şema üçlülerini hafızada tutarak gerçekleştirilen yöntem ile hemen hemen aynı zamanda bitmektedir. Pig ise diğer çözümlere göre daha uzun sürede çıkarsama işlemini tamamlamaktadır.

VM tek disk durumda genel olarak bir performans artışı sağlandığı görülmektedir. Burada en hızlı çıkarsama işlemi indeks yöntemi ile gerçekleştirilmiştir. Hafıza yöntemi de indeks yöntemi ile hemen hemen aynı sonuçları elde etmiştir. Hive bu durumda hafıza yönteminin ve indeks yönteminin gerisinde kalmıştır. Pig ise yine diğer çözümlerin çok gerisinde kalmıştır.

Aynı çıkarsama işlemleri Freebase veri kümesi üzerinde VM tek disk kurulumu üzerinde gerçekleştirilmiştir. Bu testler sonucunda Freebase için üretilecek herhangi bir üçlü bulunmadığı gözlemlenmiştir.



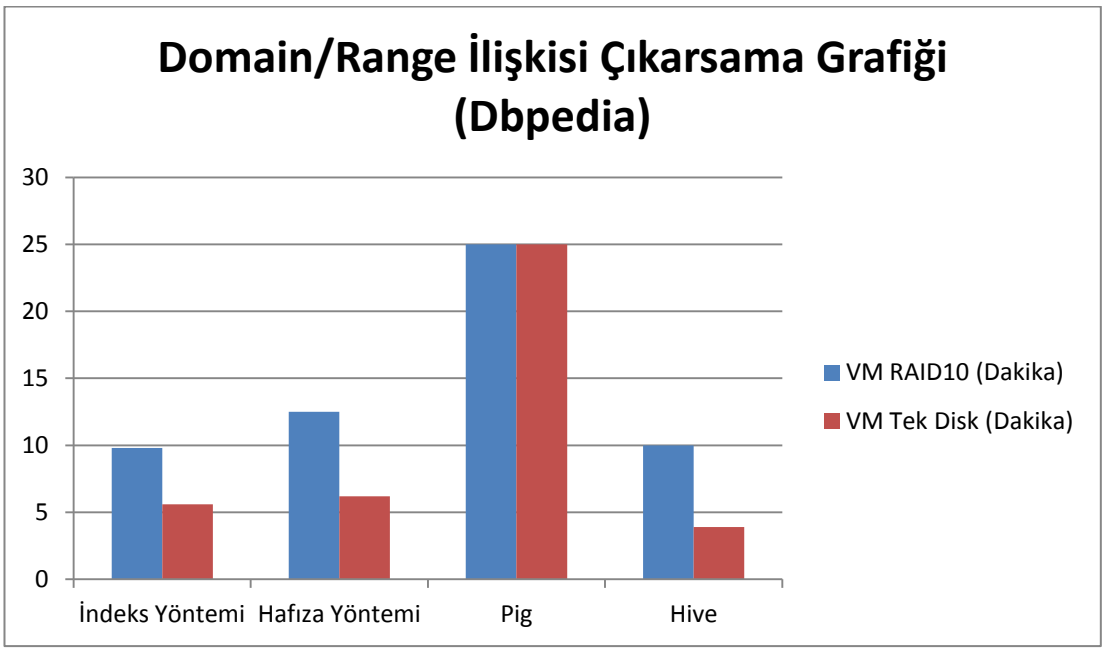
Şekil 6.2. Alt Sınıf Çıkarsama Grafiği (Freebase)

Bu grafikte görüldüğü gibi çıkarsama sonucu en hızlı Hive yöntemi ile gerçekleştirilmiştir. İndeks yöntemi ve Hafıza yöntemi birbiriyle aynı zamanda tamamlanmıştır. Pig ise diğer durumlarda olduğu gibi yine en uzun süren çıkarsama yöntemi olmuştur. Freebase'in tüm veri kümesi üzerinde bu çıkarsama işlemi gerçekleştirilmek istendiğinde de 50 dakika civarı sürmüştür.

Burada Hive ve Pig için süre hesaplaması yapılırken her bir kural için ayrı ayrı çıkarsama işlemi gerçekleştirilmiş ve daha sonra bu değerlerin toplam zamanı hesaplanmıştır.

6.2.2. Domain/Range İlişkisi için Performans Sonuçları

Domain-Range ilişkisinde çıkarsama işleminin gerçekleştirilmesi için 2 kuralın çalıştırılması gerekmektedir. Bu kuralların Dbpedia veri kümesi üzerinde çalıştırılması sonucunda elde edilen sonuçlar aşağıdaki grafikte gösterilmiştir. Bu çıkarsama işleminin sonucunda yaklaşık 5 milyon üçlü oluşmuştur.

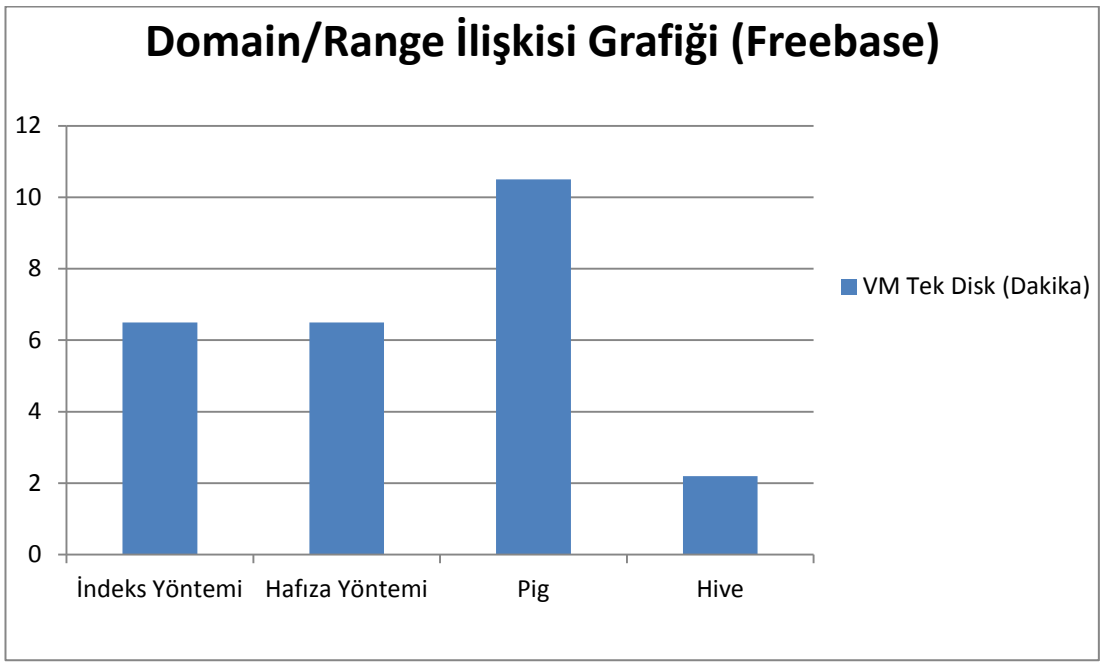


Şekil 6.3. Domain/Range İlişkisi Çıkarsama Grafiği (Dbpedia)

Burada görüldüğü gibi VM üzerindeki iki kurulumda da çıkarsama yöntemleri arasındaki başarı sıralamasında benzer sonuçlar elde edilmiştir. Çıkarsama işlemini en hızlı şekilde Hive ve İndeksleme yöntemi ile gerçekleştirilmektedir. İndeksleme yöntemi bu durumda hafızaya alma yöntemine göre performans artışı sağlamıştır bunun nedeni bu çıkarsama işleminde kullanılan şema üçlülerinin çokluğudur. Pig ile çıkarsama işlemi ise göreceli olarak oldukça uzun zaman almıştır.

VM tek disk kurulumu genel olarak çıkarsama işlemlerinde %50 'ye yakın bir performans artışı sağlamıştır. Bunun sebebi daha önce bahsedildiği gibi disk IO 'da karşılaşılan güçlüklerin bu kurulum ile azaltılmasıdır.

Aynı çıkarsama işlemleri Freebase veri kümesinde ve VM tek disk kurulumu üzerinde gerçekleştirilmiştir. Bu testler sonucunda aşağıdaki grafikteki sonuçlar elde edilmiştir.



Şekil 6.4. Domain/Range İlişkisi Grafiği (Freebase)

Bu grafikte görüldüğü çıkarsama sonucu en hızlı Hive yöntemi ile gerçekleştirilmiştir. İndeks yöntemi ve Hafıza yöntemi birbiriyle aynı zamanda tamamlanmıştır. Pig ise diğer durumlarda olduğu gibi yine en uzun süren çıkarsama yöntemi olmuştur. Freebase'in tüm veri kümesi üzerinde bu çıkarsama işlemi gerçekleştirilmek istendiğinde yaklaşık 10 saat sürmüştür ve sonucunda 32 milyon üçlü oluşmuştur.

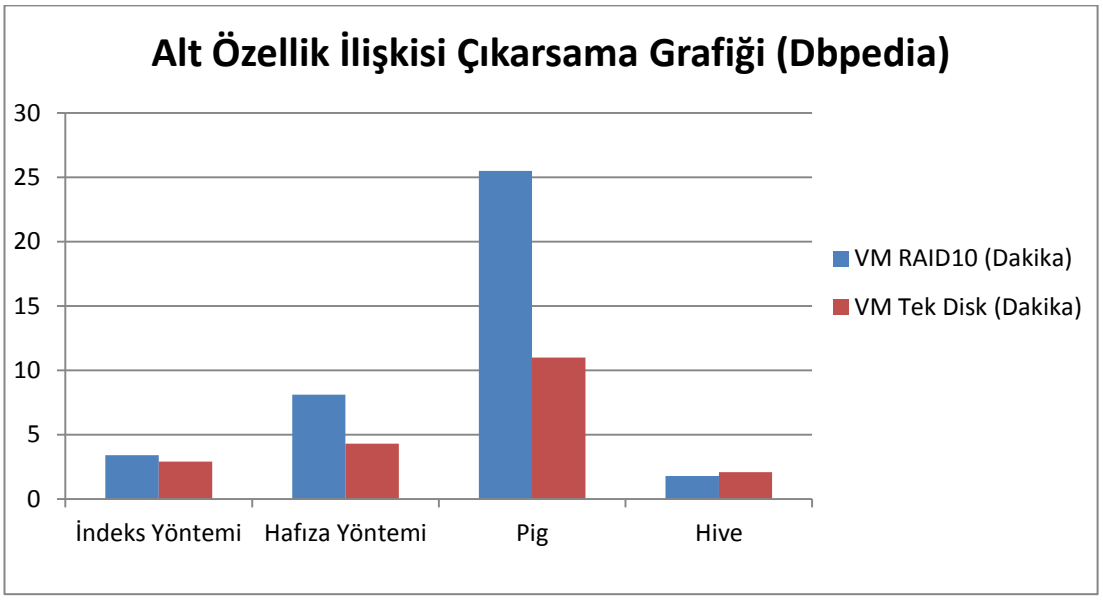
Hive ve Pig için zamanlar hesaplanırken iki kuralın çalışma süreleri toplanmıştır.

6.2.3. Alt Özellik İlişkisi İçin Performans Sonuçları

Alt özellik ilişkisinde çıkarsama işleminin gerçekleştirilmesi için 2 kuralın çalıştırılması gerekmektedir. Bu kuralların Dbpedia veri kümesi üzerinde çalıştırılması sonucunda elde edilen sonuçlar aşağıdaki grafikte gösterilmiştir. Bu çıkarsama işleminin sonucunda yaklaşık 2 milyon 300 bin üçlü oluşmuştur.

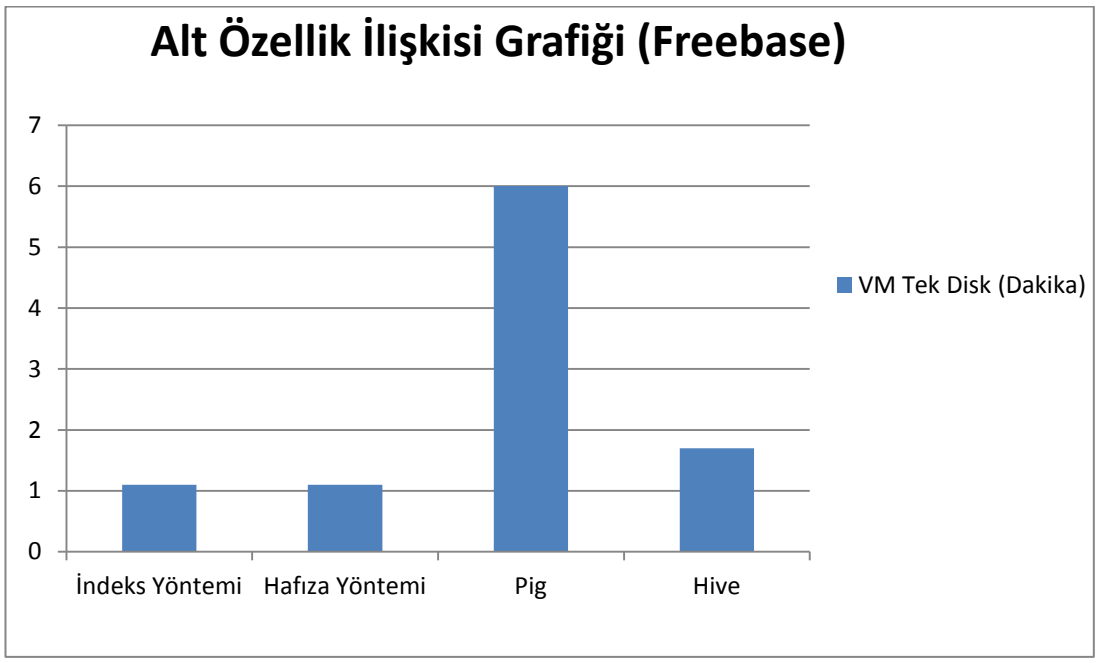
Bu grafikte görüldüğü gibi çıkarsama işlemi iki VM kurulumu içinde yöntemler arasında benzer başarı sonuçları ortaya çıkmaktadır. Daha önceki durumlarda olduğu gibi çıkarsama işlemleri Hive ile en hızlı şekilde

gerçekleştirilmiştir. Diğer performanslara bakacak olursak; indeksleme yöntemi hafızaya alma yöntemine göre bir performans artışı sağlamıştır. Ayrıca Pig'de çıkarsama işlemi oldukça uzun süre almıştır. VM 'deki tek disk kurulumu da daha önceki çıkarsama işlemlerinde olduğu gibi önemli bir performans artışı sağlamıştır.



Şekil 6.5. Alt Özellik İlişkisi Çıkarsama Grafiği (Dbpedia)

Aynı çıkarsama işlemleri Freebase veri kümesi üzerinde VM tek disk kurulumu üzerinde gerçekleştirilmiştir. Bu çıkarsama işlemi sonucunda herhangi bir üçlü oluşmamıştır.

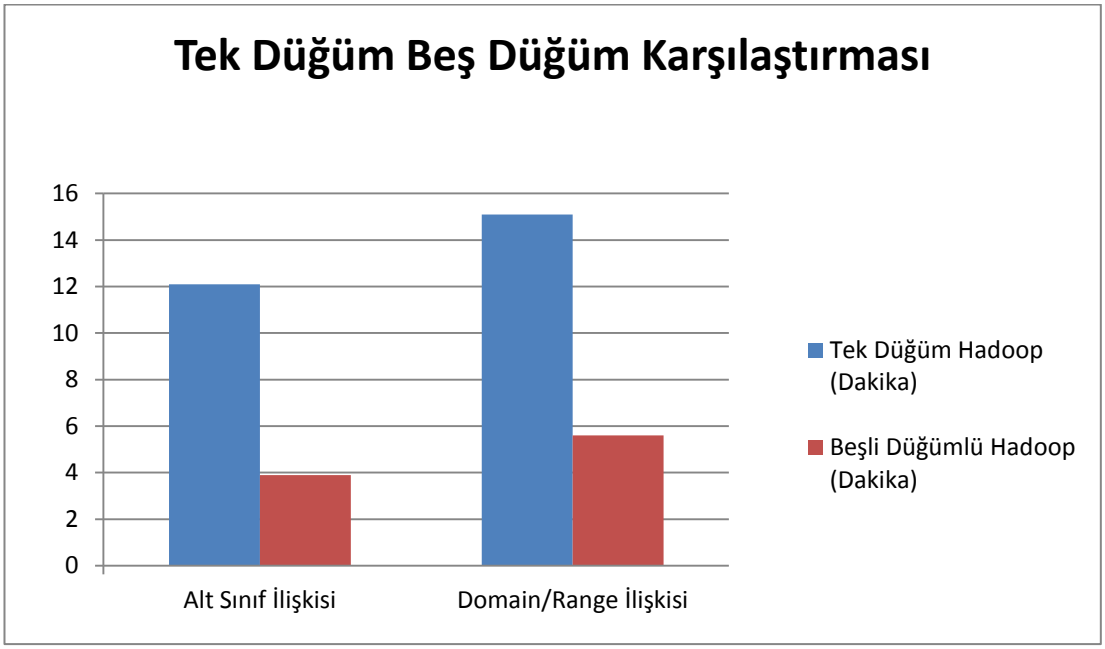


Şekil 6.6. Alt Özellik İlişkisi Grafiği (Freebase)

Bu grafikte görüldüğü gibi indeks yöntemi ve hafızaya alma yöntemi birbirlerine çok yakın sürelerle tamamlanmış ve en iyi sonuçları elde etmişlerdir. Hive yöntemi ile çıkarsama işlemi bu yöntemlere kıyasla biraz daha fazla süre almıştır. Pig ile diğer yöntemlerde olduğu gibi en uzun sürede çıkarsama işlemini gerçekleştirmiştir. Freebase'in tüm veri kümesi üzerinde bu çıkarsama işlemi gerçekleştirilmek istendiğinde yaklaşık 50 dakika sürmüştür ve herhangi bir üçlü oluşmamıştır.

6.2.4. Tekli Düğüm ile Beşli Düğüm Arasındaki Performans Farkı

Hadoop üzerinde yaptığımız testleri iki Hadoop kurulumu için yaptığımızı daha önceki bölümde belirtmiştik. Bu iki Hadoop kümesi üzerinde alt sınıf ilişkisinin ve Domain Range ilişkisinin indeksli halinin performans karşılaşması aşağıdaki grafikte görülmektedir. Bu görevler aynı algoritma ve aynı veri kümesi üzerinde çalıştırılmıştır.



Şekil 6.7. Tek Düğüm - Beş Düğüm Karşılaştırması

Burada görüldüğü gibi düğüm sayısı artırınca çıkarsama işleminin süresi azalmaktadır. Grafikte yüzde 70 oranına yakın performans artışı sağlandığı görülmektedir.

7. SONUÇ

Bu tez kapsamında RDFS çıkarsama işlemi MapReduce ve Hadoop teknolojileri ile gerçekleştirilmiştir. Burada çıkarsama işlemi 4 farklı şekilde gerçekleştirilmiştir. Birinci durumda şema üçlüleri indekslenmiş ve buradan görevlere girdi olarak kullanılmıştır. İkinci durumda şema üçlüleri hafızaya alınmış ve buradan hızlı bir şekilde görevlere girdi olarak alınmıştır. Üçüncü durumda Pig yapısı kullanılarak RDFS çıkarsama işlemi gerçekleştirilmiştir. Son olarak dördüncü durumda da Hive üzerinde çıkarsama işlemleri gerçekleştirilmiştir.

Bu çalışmada indeksleme yöntemi ile RDFS çıkarsama işlemi ilk kez denenmiştir. Hafızaya alınarak yapılan işleme göre bazı durumlarda performansta artışlar sağlanmıştır. Ayrıca bazı veri kümelerinde şema üçlülerini ayırarak hafızaya almak imkânsız olmaktadır (Örneğin Freebase). Çünkü bu veri kümelerinde tüm üçlüler tek bir dosya da saklanmaktadır. Bu dosyaları parçalara ayırıp hafızaya alınabilecek uygun konuma getirmek ayrı bir süreç ve zaman almaktadır. Öte yandan indeksleme yönteminde önceden çalıştırılan bir MapReduce görevi ile şema üçlüleri indekslenebilmekte bu da çıkarsama işleminde kolaylık sağlamaktadır.

Pig ve Hive ile RDFS çıkarsama işlemi de ilk kez bu çalışma kapsamında gerçekleştirilmiştir. Özellikle Hive ile çıkarsama işleminde önemli bir performans artışı sağlanmıştır. Burada Hive'in sağlamış olduğu 'Bucket' yapısı bu performans artışının sebebi olarak gözlemlenmiştir. Ancak Hive'in düşük yoğunluklu disk IO işlemi gerektiren çıkarsamalarda indeks ve hafızaya alma yöntemlerinin gerisinde kalabileceği de görülmüştür.

Bu test işlemleri gerçekleştirilirken kullanılan Hadoop sunucuları VM üzerinde kurulmuştur. Burada da VM üzerindeki ayarlamalara göre önemli performans farklılıkları gözlemlenmiştir. Özellikle disk IO performansında önemli performans düşüşleri olabileceği görülmüştür. Buna karşın her makinaya tek bir fiziksel disk atandığından özellikle MapReduce "shuffle" aşamasında performans artışı sağlanabildiği görülmüştür.

Tüm çalışma dikkate alındığında RDFS çıkarsama işleminin farklı MapReduce algoritmaları ve MapReduce teknolojileri kullanılarak gerçekleştirilebileceği görülmüştür. Bazı durumlarda indeksleme yöntemi ile çıkarsama işleminin performans artışı ve bazı veri kümelerinde kolay çıkarsama yapabilmeye olanak sağlayabileceği gözlemlenmiştir. Performans olarak ise bazı durumlarda Hive kullanmanın RDFS çıkarsama için önemli bir performans artışı sağlayabileceği görülmüştür. Ancak Hive ve Pig' de iç içe çıkarsama işleminin yapılabilmesi için verilen çözümlerin geliştirilmesi gerekmektedir.

KAYNAKLAR

[1] Urbani, J., Kotoulas, S., Oren, E., & Van Harmelen, F. (2009). Scalable distributed reasoning using mapreduce. In *The Semantic Web-ISWC 2009* (pp. 634-649). Springer Berlin Heidelberg.

[2] "Solr" erişim adresi: http://lucene.apache.org/solr/, erişim tarihi: Mart 2014.

[3] "Hive" erişim adresi: http://hive.apache.org/, erişim tarihi: Mart 2014.

[4] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific american*, *284*(5), 28-37.

[5] Tom White, Hadoop The Definitive Guide, O'Reilly Media / Yahoo Press, 2012

[6] Weaver, J., & Hendler, J. A. (2009). Parallel materialization of the finite rdfs closure for hundreds of millions of triples. In *The Semantic Web-ISWC 2009* (pp. 682-697). Springer Berlin Heidelberg.

[7] "Hadoop" erişim adresi: http://hadoop.apache.org/, erişim tarihi: Mart 2014.

[8] Dean Allemang; James Hendler, Semantic Web for the Working Ontologist: Modeling in RDF, RDFS and OWL, *Morgan Kaufmann*, 2008

[9] "Pig" erişim adresi: http://pig.apache.org/, erişim tarihi: Mart 2014.

[10] "RDF Sparql" erişim adresi: http://www.w3.org/TR/rdf-sparql-query/, erişim tarihi: Mart 2014.

[11] Kara, S., Alan, Ö., Sabuncu, O., Akpınar, S., Cicekli, N. K., & Alpaslan, F. N. (2012). An ontology-based retrieval system using semantic indexing.*Information Systems*, *37*(4), 294-305.

[12] Husain, M., Khan, L., Kantarcioglu, M., & Thuraisingham, B. (2010, July). Data intensive query processing for large RDF graphs using cloud computing tools. In*Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on* (pp. 1-10). IEEE.

[13] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., ... & Murthy, R. (2010, March). Hive-a petabyte scale data warehouse using hadoop. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on* (pp. 996-1005). IEEE. [14] Alex Holmes, Hadoop in Practice, Mannig, 2012

[15] Tanimura, Y., Matono, A., Lynden, S., & Kojima, I. (2010, March). Extensions to the Pig data processing platform for scalable RDF data processing using Hadoop. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on* (pp. 251-256). IEEE.

[16] Jang, B., & Ha, Y. G. (2013, July). Transitivity Reasoning for RDF Ontology with Iterative MapReduce. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on* (pp. 232-237). IEEE.

[17] Zhanga, Y., Chenb, T., Youc, W., Yud, J., Sune, J., & Chenf, H. A New Efficient Semantic Web Platform Based on the Solr, SIREn and RDF.

8. ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, Adı	: ÇETİN, Yiğit
Uyruğu	: T.C.
Doğum tarihi ve yeri	: 13.09.1987 İzmir
Medeni hali	: Bekar
Telefon	: 0 (555) 704 11 24
E-mail	: ycetin@etu.edu.tr

Eğitim

Derece	Eğitim Birimi	Mezuniyet tarihi
Lisans	Ege Üniversitesi Bilgisayar Mühendisliği	2009
Yüksek Lisans	TOBB Ekonomi ve Teknoloji Üniversitesi Bilgisayar Mühendisliği	2014

İş Deneyimi

Yıl	Yer	Görev
2010 -	Ekinoks Yazılım	Yazılım Mühendisi

Yabancı Dil

İngilizce

Yayınlar

1. Çetin, Yiğit; Çetin, Övünç; Özkan, Beytun; Savaşçı, Mustafa; Ulusu, Onur; Ekinci, Erdem Eser; Dikenelli, Oğuz, "Etmen-Servis Tümlesimi İçin Bir Etmen Altyapısı", 4. *Ulusal Yazılım Mühendisliği Sempozyumu 2009 (UYMS)*

2. Cetin, Yigit; Abul, Osman, “Distributed RDFS Reasoning With MapReduce”, 29th International Symposium on Computer and Information Sciences 2014 (ISCIS)