EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

DAĞITIK BAĞLI VERİ SORGULAMA MOTORLARINDA PERFORMANS YÖNETİMİ

Burak YÖNYÜL

Tez Danışmanı : Doç. Dr. Rıza Cenk ERDUR

Bilgisayar Mühendisliği Anabilim Dalı

Bilim Dalı Kodu : 619.01.00
Sunuş Tarihi : 07.01.2014

Bornova-İZMİR
2014

Burak YÖNYÜL tarafından **Yüksek Lisans** tezi olarak sunulan "**DAĞITIK BAĞLI VERİ SORGULAMA MOTORLARINDA PERFORMANS YÖNETİMİ**" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve **07/01/2014** tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:	**İmza**
Jüri Başkanı : Doç. Dr. Rıza Cenk ERDUR	**.....**
Raportör Üye: Prof. Dr. Oğuz DİKENELLİ	**.....**
Üye : Yrd. Doç. Dr. Belgin ERGENÇ	**.....**

ÖZET

DAĞITIK BAĞLI VERİ SORGULAMA MOTORLARINDA PERFORMANS YÖNETİMİ

YÖNYÜL, Burak

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı
Tez Danışmanı: Doç. Dr. Rıza Cenk ERDUR
Ocak 2014, 91 sayfa

Bağlı veri ilkeleri ile veb, sayısı her geçen gün artan dağıtık ve birbirine bağlı veri kümelerinin tek bir veritabanı gibi sorgulanabileceği bir veri uzayına dönüşmüştür. Sorguyu kullanıcıların yerine, bu veri kümeleri üzerine dağıtan ve etkin şekilde sorgulayan birleştirilmiş sorgu motorları ile bu veriden çok daha iyi şekilde yararlanılabilmektedir. Bu tez kapsamında VOID üstverilerini kullanan birleştirilmiş SPARQL sorgulama motoru olan WoDQA üzerinde gerçekleştirilen performans iyileştirme yöntemleri anlatılmaktadır. WoDQA birbirine bağlı veri kümelerinin üstverisini ve üçlü desenleri arasındaki ilişkileri, veri kümesi seçim yöntemine dahil eder. Tez kapsamında WoDQA'nın oluşturduğu birleştirilmiş sorgunun eniyileştirilmesi için sezgisel yöntemler kullanılmıştır ve eniyileştirilen bu sorgunun etkin ve hızlı bir şekilde çalıştırılıp sonuçların da hızlıca elde edilmesi için SPARQL FILTER tabanlı bağlı birleştirme yöntemi uygulanmıştır. Bu yaklaşımlar FedBench değerlendirme kütüphanesi ile değerlendirilmiş ve WoDQA'nın seçim ve işletim yaklaşımları üzerinde olumlu etkisi ortaya konmuştur. Ayrıca birleştirilmiş SPARQL sorgu doğasına uygun bir SPARQL önbellekleme mimarisi gerçekleştirilmiştir. Mimaride SPARQL sorgularının cevapları üçlüler olarak, her bir üçlünün bellekte yalnızca bir kez yer alacağı şekilde tutulmaktadır. Böylece önbellek etkin olarak yönetilerek sorgular hızlıca çalıştırılabilmektedir. Önbellek mimarisi tekil ve birleştirilmiş SPARQL sorguları ile deneysel olarak değerlendirilmiş ve WoDQA'nın performansına olan iyileştirici etkisi ortaya konmuştur.

Anahtar sözcükler: Dağıtık Sorgulama, Birleştirilmiş Sorgular, Bağlı Veri, SPARQL, Anlamsal Veb, Değerlendirme, Eniyileştirme, Önbellekleme.

ABSTRACT

PERFORMANCE MANAGEMENT IN FEDERATED LINKED DATA QUERY ENGINES

YÖNYÜL, Burak

MSc in Computer Engineering
Supervisor: Assoc. Prof. Dr. Rıza Cenk ERDUR
January 2014, 91 pages

With the linked data principles web evolved into a dataspace in which increasing number of datasets can be queried as a single database. Using the federated SPARQL query engines, which federate queries on datasets and can query effectively in spite of users, it is possible to benefit much beter from this data. In this thesis, performance improvements on WoDQA, which is a linked data query engine that depends on VOID metadata, will be presented. WoDQA incorporates interlinking metadata between datasets and the relations between triple patterns of a query into dataset selection. Within the thesis, some heuristics for query optimization and a SPARQL FILTER keyword based bound join implementation for query execution are employed. The performance of these approaches are evaluated using FedBench suite, and the improvement provided by the selection and execution approach of WoDQA is shown. Besides a SPARQL caching architecture that fits perfectly to the nature of federated SPARQL queries is implemented. In the architecture, the results of SPARQL queries are stored as triples and somehow each triple must reside in memory only once. Thus queries can be executed faster while cache is managed efficiently. The SPARQL cache approach is evaluated with single and federated SPARQL queries, and its improvement to the performance of WoDQA is presented.

Keywords: Fedeated Querying, Federated Queries, Linked Data, SPARQL, Semantic Web, Evaluation, Optimization, Caching.

TEŞEKKÜR

Eğitimim ve tez çalışmam sürecinde bana verdikleri destekten dolayı Doç. Dr. Rıza Cenk Erdur, Prof. Dr. Oğuz Dikenelli ve Erdem Eser Ekinci'ye, her durumda eğitimimi destekleyen ve manevi desteğini esirgemeyen aileme, arkadaşım Ramadan Altınkaya'ya, bana sağladıkları çalışma ortamı için Tayfun Gökmen Halaç, Zehra Gül Çabuk, Ziya Akar, Pınar Göçebe, Çağdaş Olgun, Gürkan Şafak, Enes Bulut, Bahtiyar Erden, Emrah İnan, Damla Oğuz, Berkay Akdal, Tolga Takır, Tunahan Özkan ve Ceren Tüker'e ve "111E027" numaralı "Etmenler İçin Anlamsal Ortam Tasarımı ve Gerçekleştirimi" isimli proje kapsamında yüksek lisans çalışmamı maddi olarak destekleyen Türkiye Bilimsel ve Teknolojik Araştırma Kurumu'na (TÜBİTAK) teşekkürü bir borç bilirim.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

İÇİNDEKİLER (devam)

Sayfa

ŞEKİLLER DİZİNİ

ŞEKİLLER DİZİNİ (devam)

Şekil Sayfa

ÇİZELGELER DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler	Açıklama

Kısaltmalar

aQEToa	Average Query Execution Time Over All Queries
aQET	Average Query Execution Time
BGP	Basic Graph Pattern
cQET	Composite Query Execution Time
FIFO	First In First Out
IRI	Internationalized Resource Identifiers
LFU	Least Frequently Used
LRU	Least Recently Used
maxQET	Maximum Query Execution Time
minQET	Minimum Query Execution Time
oaRT	Overall Runtime
QMpH	Queries Mixes per Hour
QpS	Queries per Second
RDF	Resource Description Framework
TTI	Time to Idle

SİMGELER VE KISALTMALAR DİZİNİ (devam)

Kısaltmalar	Açıklama
TTL	Time to Live
URI	Uniform Resource Identifier
VOID	Vocabulary of Interlinked Datasets

1 GİRİŞ

Bağlı veri standartları sayesinde veb, yapısal olmayan belgelerin ağı olmaktan çıkıp yapısal verinin bir ağı haline gelmiştir. Bu gelişim, yapısal veri kümelerinin yayımlanması ve veb üzerinde bu veri kümeleri arasında bağların tanımlanmasıyla gerçekleşmiştir. Veri vebi üzerindeki dağıtık ve birbirine bağlı veri kümelerini sorgulamak, veriyi elde etmenin önemli bir yolu haline gelmiştir. Bu yöntem, bağlı verinin kurumiçi ve kurumlar arası veri bütünleştirmesi için kullanılmasında ve dağıtık veri kümelerinden oluşan veri uzayını sorgulayan bilgi sistemlerinin geliştirilmesinde ümit verici olarak görülmektedir. Veri kümelerinin çalışma zamanında eklenip çıkarılabildiği açık ve değişken veri uzaylarının oluşturulması bağlı veri standartları ile mümkün hale gelmiştir. Bununla birlikte bir sorguyu dağıtık veri kümeleri üzerinde işletmek için ilişkili veri kümelerinin otomatik olarak keşfedilmesi ve bu veri kümelerinden gelen sonuçların da etkin bir şekilde birleştirilmesi gerekmektedir. Bu bağlamda sorguyu bizim yerimize ilişkili veri kümeleri üzerine dağıtan ve buradan dönen sonuçları birleştirip nihai sonucu bize döndüren birleştirilmiş SPARQL sorgu motorları geliştirilmişir.

Temelde bir SPARQL sorgu motorundan beklenen iki ana hizmet vardır. Bunlardan ilki tam sonuç döndürme, ikincisi ise bu sonuçları olabildiğince hızlı elde etmedir. Tam sonuç döndürmenin zaten önemli bir zorunluluk olduğu düşünülürse, bir sorgu motorunun performansı, diğerlerine karşı olan üstünlüğünü ve farkını ortaya koymaktadır. Hızın neredeyse her alanda çok önemli bir etken olduğu düşünülürse, bunun birleştirilmiş sorgulamada da kaçınılmaz bir gereklilik olduğu açıktır. Buna göre ihtiyaç duyulan bu hızı elde etmek için; birleştirilmiş SPARQL sorgu motorlarının veri kümesi seçimini etkin yapması ve alt sorguların olabildiğince ilişkili veri kümeleri üzerinde işletilmesi, sorgular için etkin işletim planı oluşturulması, alt sorgulardan dönen cevapların en az sayıda HTTP isteği oluşturacak şekilde birleştirilmesi, önbellekleme yöntemi ile sorgu işletimi maliyetinin ortadan kaldırılması gibi yöntemlerin uygulanması da birer gereklilik olmaktadır. Bu tez kapsamında birleştirilmiş bağlı veri sorgu motoru olan WoDQA (Akar vd., 2012) (Web of Data Query Analyzer - Veri Vebi Sorgu Çözümleyicisi)[1] tanıtılacak, WoDQA üzerinde gerçekleştirilen performans eniyileştirmelerinden bahsedilecek ve bu eniyileştirmelerin etkisinin deneysel çalışmalar ile ölçümleri ortaya konacaktır. Daha sonra, gerçekleştirilen SPARQL önbellekleme mimarisinin yapısı ve

[1] WoDQA'nın son sürümü buradan indirilebilir: http://seagent.ege.edu.tr/etmen/wodqa.html

işleyişi anlatılacak, bu önbellek mimarisinin tek uçnoktalı ve birleştirilmiş sorgular üzerindeki etkisi ayrı ayrı değerlendirilecek ve önbellek yaklaşımının WoDQA'nın performansına olan olumlu etkisi ortaya konacaktır. Ayrıca, WoDQA'nın gerçekleştirilen eniyileştirmeler ile değerlendirilmesiyle bu bağlamda bir yayın çalışması (Halaç vd., 2013) da yapılmıştır.

WoDQA sorgu motoru veri kümelerinin üstverilerinin yer aldığı VOID[2](Vocabulary of Interlinked Datasets) (Alexander et al., 2009) belgelerini kullanarak ilişkili veri kümelerini belirlemekte ve sorguyu bu dağıtık veri kümeleri üzerinde işletmektedir. WoDQA birleştirilmiş sorgulamayı üç adımda uygular: veri kümesi seçimi, sorgu planı eniyileştirmesi ve birleştirim eniyileştirmesi. WoDQA bir sorgunun ilişkili veri kümelerine karar verirken bunu, gereksiz sorgulamadan kaçınmak için olabildiğince en ilişkili veri kümelerini seçerek yapmayı ve sorgunun sonucunu eksiksiz döndürmeyi amaçlar. Veri kümesi seçimi iki yöntemden yararlanılarak gerçekleştirilir. İlk yöntem veri kümesi ve bağ kümesi tanımlarının yer aldığı VOID tanımları kümesidir. İkinci yöntem ise sorgu yapısıdır, örn. üçlü desenleri arasındaki ilişkiler. Üçlü desenleri arasındaki ilişkiler VOID tanımları ile birleşince bir sorgu için tamamen ilişkisiz veri kümelerini belirlemek mümkün olmaktadır. İlişkili veri kümelerinin belirlenmesinden sonra WoDQA, aynı veri kümesinden sorgulanacak üçlüler için harici grupları belirleme ve grup içerisindeki üçlülerin ve üçlü gruplarının yerlerini değiştirerek etkin şekilde çalışmasını sağlama gibi eniyileştirmeler uygulamaktadır. Harici gruplar işletim sırasında yollanan HTTP isteği sayısını azaltmaktadır ve yer değiştirme işlemi ise harici bir grubun işletilmesini hızlandırmayı ve ara sonuç sayısını azaltmayı sağlamaktadır. Son olarak WoDQA alt sorguları dağıtmak için bağlı birleştirme stratejisini kullanır. Böylece uzak veri kaynakları üzerinde birleştirme hesapları yapılır. Bu yaklaşımın katkılarından aşağıda bahsedilmektedir.

- Veri kümeleri arasındaki bağlantılar ve üçlü desenleri arasındaki ilişkiler veri kümesi seçimine dahil edilmiştir. Böylece ilişkili veri kümeleri daha etkin bir şekilde seçilebilmektedir.

- SPARQL FILTER kullanılarak bağlı birleştirme yöntemi gerçekleştirilmiştir. IN ve OR ifadeli FILTER kullanımları tecrübe edilmiş ve gelişmiş bir RDF sunucusu olan 4Store üzerinde FILTER-OR kullanımı ile gerçekleştirilen bağlı birleştirme yaklaşımı en iyi performansı göstermiştir.

[2]http://www.w3.org/TR/v

- FedBench değerlendirmesine göre, WoDQA'da uygulanan bu yaklaşımların, eksiksiz sonuç elde etmeyi hedefleyen diğer gelişmiş sorgu motorlarından (Schwarte et al., 2011a; Görlitz and Staab, 2011) daha iyi olduğu gözlemlenmiştir.

Günümüzde Veri Vebi üzerinde çalışan birçok uygulama vardır ve sayısı her geçen gün artmaktadır. Bu uygulamaların büyük çoğunluğu da Veri Vebi'ni belirli sıklıkta sorgulamaktadır. Bu nedenle Veri Vebi üzerinde çalıştırılan SPARQL sorguları, genellikle uygulamalar tarafından atılmaktadır ve insanlar tarafından atılan sorgu miktarı oldukça küçük boyuttadır. Uygulamalar belirli sabit bir mantık ve otomasyon çerçevesinde çalıştığından atılan sorgular da genellikle hep aynı ve birbirine benzer olmaktadır. Sorgunun gerçek Veri Vebi üzerinde çalıştırıldığı düşünüldüğünde, verinin tutulduğu RDF depoları birbirinden farklı olduğundan, performansı değişiklik gösterdiğinden ve bu RDF depolarının dışarıya açıldığı uçnoktalar sürekli erişilebilir olmadığından, sorguyu her seferinde tekrar çalıştırmak büyük bir maliyet oluşturmaktadır. Bu sorunlardan hareketle, SPARQL sorgulamada önbellekleme kullanılmasının sorgulama performansını önemli ölçüde artıracağı açıktır. Böylece hem sorguların uçnoktalar üzerinde çalıştırılma maliyetinden kurtulunmuş olunur, hem de sorgunun sorgulandığı uçnokta(lar) erişilemez durumda olsa bile cevabı önbellekte yer alıyorsa elde edilebilir.

Tezin ilerleyen kısımlarında, gerçekleştirilen SPARQL önbellekleme mimarisi olan Bellek Çizgesi anlatılacak ve bu mimarinin SPARQL sorgulamaya ve WoDQA'ya olan katkıları deneysel ölçümlerle değerlendirilecektir. Bellek Çizgesi, bilindik önbellekleme yeteneklerine ek olarak sorgu cevaplarını, bellekte yalnızca tek bir kez yer alabilen üçlüler şeklinde tutup bir çizge yapısı oluşturarak, önbelleğin büyüklüğünü etkin şekilde yönetmiş olur. Ayrıca mantıksal olarak aynı olan sorguları belirleyerek sorguları tek tip hale getirir ve önbellekte aynı sorguların tutulmasını engelleyerek gereksiz yere işletim maliyetinden kaçınılmış olunur.

Tezin devamı şu şekilde düzenlenmiştir. Bölüm 2'de ilgili çalışmalardan bahsedilmektedir. Bölüm 3, genel WoDQA mimarisini ve gerçekleştirilen eniyileştirmeleri anlatmaktadır. Bölüm 4'te, WoDQA'nın uygulanan eniyileştirmeler ile performans değerlendirmesi yapılmıştır. Bölüm 5'te gerçekleştirilen SPARQL önbellekleme mimarisi olan Bellek Çizgesi yaklaşımı anlatılmaktadır. Bölüm 6'da Bellek Çizgesi'nin tek uçnoktalı sorgularda ve birleştirilmiş sorgulamadaki deneysel değerlendirmesi ve Bellek Çizgesi yaklaşımı dahil edildiğinde WoDQA'nın performansına olan etkisi gösterilmiştir. Son olarak Bölüm 7'de sonuç ortaya konmaktadır.

2 İLGİLİ ÇALIŞMALAR

Bağlı veri sorgu işleme yaklaşımları merkezi ve dağıtık olmak üzere iki sınıfa ayrılır. Merkezi sorgulama yaklaşımında bağlı veri tek bir merkezi veri deposuna toplanır ve veri bu depodan sorgulanır. Bu yaklaşım, önceden seçilen veri kaynaklarının toplandığı veri ambarcılığını ve Veb'in gezilip RDF bağlarının dolaşılarak, keşfedilen verinin (Hartig and Langegger, 2010) dizinlendiği arama motorlarını da içermektedir. Ancak bu yaklaşımın ana olumsuz yanı sorgulanan verinin canlı olmaması, yani asıl kaynakların kopyası olmasıdır.

Diğer yandan dağıtık sorgulamada, sorgu parçaları doğrudan asıl veri kaynaklarından sorgulanır ve elde edilen dağıtık haldeki veri birleştirilir. Sorgu birleştirme (Görlitz and Staab, 2011; Haase et al., 2010) ve burnunu-takip-et (Hartig et al., 2009) yöntemleri başlıca dağıtık sorgulama yaklaşımlarıdır. İlk yaklaşım olan sorgu birleştirmenin DARQ (Quilitz and Leser, 2008), FedX (Schwarte et al., 2011a) ve SPLENDID (Görlitz and Staab, 2011) gibi çeşitli uygulamaları mevcuttur. DARQ sorguyu, geliştirici tarafından oluşturulan ve Servis Tanımları [1] olarak adlandırılan veri kümesi üstverisine göre dağıtır ve üçlü ile varlık sayısından yararlanarak sorgu planını eniyileştirir. DARQ ilişkili veri kümelerini seçmek için yüklemleri kullanır, bundan dolayı sorgu işletiminin başarısı üçlüleri yüklemler ile ilişkilendirmeye bağlıdır ve bu durumda bağlantısız yüklemler ele alınamamış olur[2]. Diğer yandan FedX'ten bahsedilecek olursa Sesame'nin Birleştirme SAIL yönteminin bir uzantısıdır. FedX her üçlü için ilişkili veri kümelerine, sırayla bu üçlülerin tüm veri kümeleri üzerindeki varlığını ASK sorguları ile sorgulayarak karar verir (Schwarte et al., 2011b). Bu iki araç standart bir veri kümesi tanımlama yaklaşımı kullanmamaktadır. Halbuki bağlı verinin kendini-tanımlama doğasında öngörüldüğü gibi, veri kümelerinin üstverisinin yayımlayıcılar tarafından tanımlandığı, VOID tanımlamaları gibi standart bir üstveri kullanmak oldukça yararlı olacaktır. Son sorgu birleştirme gerçekleştirimi SPLENDID ise VOID tanımlamalarını kullanmaktadır. Veri kümelerini VOID tanımlamalarındaki istatistiksel veriye göre dizinler, bu dizinden ve ASK sorgularından yararlanarak veri kümelerini seçer ve birleştirme düzenini de VOID'teki istatistiksel veriye göre eniyi-

[1] Belirtilen Servis Tanımları; veri kümesinde yer alan üçlüler ile ilgili bilgiler, erişim desenlerindeki kısıtlamalar ve veri kümeleri hakkında istatistiksel bilgiler içermktedir.

[2] http://darq.sourceforge.net/#Limitations_and_known_issues

leştirir.

Birleştirilmiş yaklaşımı kullanan bu araçlarda uygulanan eniyileştirme yöntemleri mantıksal ve fiziksel eniyileştirmeler olarak iki kısıma ayrılabilir. Mantıksal eniyileştirmeler kapsamında ilk yöntem sorgu planının belirlenmesidir. Buna göre uçnoktaya atılacak sorgudaki üçlü desenleri, az maliyetli olan üçlü desenleri daha önce işletilecek şekilde sıralandığında, sorgu uçnokta üzerinde daha hızlı çalıştırılabilmektedir. Sorgu planının eniyileştirmesi DARQ, FedX, SPLENDID araçlarının hepsi tarafından gerçekleştirilmektedir. İkinci mantıksal eniyileştirme yöntemi, harici grupların belirlenmesi işlemidir. Aynı uçnoktaya sorulacak üçlüler gruplanarak, uçnoktaya daha az HTTP isteği gönderilmesi sağlanmış olur. Adı geçen üç araçtan, FedX ve SPLENDID harici grupları belirleme yaklaşımını uygulamaktadır. WoDQA'da sorgu planı eniyileştirmesi ve harici grupların belirlenmesi sezgisel yöntemler kullanılarak gerçekleştirilmiştir. Fiziksel eniyileştirme ise birleştirme stratejisinin eniyileştirmesi olarak düşünülebilir. Yuvalanmış döngüsel birleştirmede (nested loop join) bir ara sorgunun sonuçları elde edildikten sonra cevap eşleşmelerinin her biri, bir sonraki ara sorguda yerine konularak yeni ara sonuçlar elde edilir. Bu birleştirme yönteminde her ara sonuç için yeni bir HTTP isteği yollamak ara sonuç sayısı fazla olan sorgularda ağ maliyeti oluşturmaktadır. Bağlı birleştirme yönteminde ise dönen arasonuçların hepsi FILTER , UNION, VALUES vb. yöntemler ile topluca birleştirilerek ara sorgularda atılan HTTP isteği sayısının düşürülmesi ve ağ maliyetinin azaltılması sağlanır. Hash birleştirme (hash join) yönteminde ise ara sorguların her biri paralel olarak ilgili uçnoktalara gönderilir ve dönen ara sonuçlar birleştirilerek ana sonuç elde edilir. Hızlı sorgu işletimi bakımından, bağlı birleştirme ve hash birleştirme yöntemleri, yuvalanmış döngüsel birleştirme yaklaşımına göre daha iyi performans göstermektedir. Bu iki yöntem kıyaslanacak olursa, hash birleştirme yöntemi ara sonuç sayısının az olduğu sorgularda etkin olarak çalışmakta, büyük ara sonuç içeren sorgularda performansı kaybolmaktadır. Buna karşılık bağlı birleştirme yöntemi ara sonuç sayısının hem az hem de fazla olduğu sorgularda oldukça iyi performans sergilemektedir. İşletim yönünden bakılacak olursa, DARQ, erişim desenlerindeki kısıtlamalara göre yuvalanmış-döngüsel-birleştirme ve bağlı birleştirmeyi destekler, FedX bağlı birleştirme stratejisini uygular ve SPLENDID ise işletim planının tahminlenen maliyetine göre hash birleştirme veya bağlı birleştirme kullanır. WoDQA'da bağlı birleştirme yöntemi uygulanmıştır ve yuvalanmış döngüsel birleştirmeyi de desteklemektedir. WoDQA, üzerinde gerçekleştirilen

tüm bu mantıksal ve fiziksel eniyileştirmeler ile değerlendirilmiş ve performans olarak bu araçların tümünü geride bırakmıştır.

Bahsi geçen bu araçlar bağlı veri kümelerini sorgulamayı amaçlarlar ancak veri kümesi seçiliminde verideki bağları hesaba katmazlar. WoDQA'da VOID'teki bağ kümesi yaklaşımı veri kümesi seçimine dahil edilmiş ve böylece birbirine bağlı veri kümelerinin tüm yapısını hesaba katarak veri kümesi seçimi geliştirilmiştir. Diğer uygulamaların bunun dışında bazı eksik yönleri de mevcuttur. İlki, ilişkili veri kümelerine karar verirken yalnızca yüklem indeksi kullanıldığında yüklemi değişken olan bağsız yüklemler veya `owl:sameAs`[3], `foaf:page`, vb. gibi tüm veri kümelerinde çokça bulunan yüklemlerin olduğu durumlarda veri kümesi seçimi etkin şekilde yapılamaz. Olası çözümlerden biri ise üçlü desenleri arasındaki ilişkileri veri kümesi seçimine dahil etmektir. SPLENDID'in sameAs desenlerini gruplama yöntemi bu durum için basit bir örnek olarak gösterilebilir. Montoya ve diğerleri (Montoya et al., 2012a) üçlü desenleri arasındaki ilişkileri veri kümesi seçimine dahil etmektedirler, fakat mevcut diğer çalışmaların aksine bu çalışmada sonucun eksiksiz bir şekilde elde edilmesi garanti edilmemektedir. İkinci eksik yön ise eğer üçlüler için birçok veri kümesi ilişkili olarak seçildiyse ASK sorguları önemli bir maliyet oluşturmaktadır. Bu nedenle ASK sorgularından önce ilişkisiz veri kümelerini elemek soğuk-önbellek performansını artırmaktadır.

Bir diğer dağıtık sorgulama yaklaşımı burnunu-takip-et (Hartig et al., 2009) yöntemi, sorgu işletimi sırasında verideki RDF bağlarını dolaşarak ilişkili veriyi keşfetmeyi amaçlar. Sorgu birleştirmedeki gibi veri kümeleri hakkında öntanımlı herhangi bir üstveriye ihtiyaç yoktur, ancak burnunu-takip-et yönteminin veri kümelerini keşfetmeye başlayabilmesi için bazı üçlüler içerisinden başlangıç URIlerinin seçilmesi gerekmektedir. Bu yaklaşımın ana olumsuz yanları; sonsuz bağ keşfi ihtimali, büyük RDF çizgelerini getirme ihtimali, sadece bağlı yükleme sahip ifadeler (`?s :prop ?o`) ve tipli ifadeler (`?s rdf:type :C`) gibi durumlarda ilişkili verileri keşfedememe gibi durumlardır. Bu kısıtlamalar daha az kapsamlı sonuç kümelerine neden olmaktadır. Burnunu-takip-et yaklaşımlarından çok bilinen bir tanesi SQUIN'dir (Hartig et al., 2009). SQUIN işletim maliyetini azaltmak ve daha kapsamlı sonuçlar sağlamak için bazı sezgisel yaklaşımlar kullanılarak geliştirilmiş olsa da (Hartig, 2011), sonuçlar başlangıç noktasına ve işletim düzenine sıkı sıkıya bağlıdır. Diğer yandan, Bouquet

[3]Tezin anlatımında kullanılan tüm ön-ekler Tablo 4.5'te tanımlanmıştır.

ve diğerleri veri vebini biçimselleştirmişler ve ilişkili çizgeleri birleştirip sorguları bu birleşmiş çizgeler üzerinde çalıştıran üç farklı sorgulama yöntemi önermişlerdir (Bouquet et al., 2009). Bu yöntemlerden birisi sorgu işletiminden önce URIlere bakarak ilişkili çizgeleri bulup birleştiren burnunu-takip-et yöntemini kullanmaktadır.

WoDQA bir sorguyu, birbirine bağlı veri kümelerinin ağı üzerine VOID üstverisini kullanarak dağıtmayı hedefler. Veri kümesi ve bağ kümesi tanımlarına ve üçlü deseni ilişkilerine bağlı bir kurallar demetiyle ilişkisiz veri kümelerini bulmayı ve onları elemeyi amaçlar. Daha sonra dağıtık sorguyu işletmek için FILTER tabanlı bağlı birleştirme yöntemi uygular. Diğer gelişmiş ve eksiksiz sonuç bulmayı hedefleyen dağıtık sorgu motorları, bağ kümesi tanımlarını ve detaylı üçlü deseni ilişkilerini veri kümesi seçimine dahil etmemektedir. Ölçümler sonucunda, WoDQA'nın sorgu çalıştırma stratejisinin diğer dağıtık sorgu motorlarından çok daha performanslı sonuçlar verdiği gösterilmiştir.

Veri Vebi'ni sorgulayan araçların sayısı arttıkça, SPARQL önbellekleme de sorgulama performansını artırmak için mükemmel bir yöntem haline gelmiştir. Arama motorları, veritabanı uygulamaları gibi alanlarda önbellekleme oldukça yaygın olarak kullanılmakla beraber, bağlı verinin yaygınlaşmaya başlamasıyla SPARQL önbellekleme ile ilgili çeşitli çalışmalar yapılmaya başlanmıştır. İlk değinilecek çalışmada (Williams and Weaver, 2011), SPARQL sorgularının önbellekte yer alma süresini HTTP tabanlı belirlemeye odaklanmıştır. Buna göre uçnoktadan sorgu cevabı elde edilirken, gelen HTTP cevabının başlığında yer alan sorgu cevabının değişim tarihine ve değişip değişmeme durumuna bakılır. Geliştirdikleri algoritma ile her sorgu için HTTP cevap başlığındaki bu değerlerden sorgunun cevabının önbellekte yer alma süresi belirlenir. Böylece önbellekte değişmeyen sorguların cevabı daha uzun süre tutulmakta ve değişken sorgu cevapları ise tahliye edilmektedir. Bu yaklaşımda önbellek bakımını etkin şekilde yapmak amaçlanmıştır. Geliştirdikleri yaklaşımın etkinliği, uçnokta değerlendirme kütüphanesi olan Berlin SPARQL Kıyaslama Kütüphanesi (Bizer and Schultz, 2009) genişletilip, Pareto Dağılımı uyarlanarak ölçülmüştür.

Bir diğer çalışmada (Martin et al., 2010), SPARQL sorgularını çözümleyerek üçlü deposu güncellemelerine göre önbelleği yöneten ve önbellekleme yaklaşımına, bağımlılık izlemeyi kullanarak birleşik uygulama nesnelerini de dahil eden bir yöntem sunmaktadır. Gerçekleştirdikleri RDF sorgu önbellekleme yaklaşımı anlamsal veb uygulaması ile SPARQL uçnokta arasında çalışmaktadır. Sorgu cevaplarındaki nesnelerin

yalnızca bir kere önbelleğe alınmasını sağlayarak önbelleği etkin yönetmeyi amaçlamışlardır. Berlin SPARQL Kıyaslama Kütüphanesi'ni genişletip Pareto Dağılımı'nı uyarlamışlar ve böylece önbelleklemeye daha uygun sorgular üretmeyi sağlamışlardır. Daha sonra kütüphanenin e-ticaret kullanım durumu güncellemeli olarak çalıştırarak gerçekleştirdikleri SPARQL önbelleklemenin performansını ölçülmüşler ve son olarak Vakantieland[4] isimli Anlamsal Veb uygulaması üzerinde, önbellek mimarisinin olumlu etkilerini sınamışlardır.

Bu bölümde değinilecek son çalışmada (Johannes Lorey, 2013), SPARQL sorgularının genellikle uygulamalar tarafından atıldığı ve bu sebeple de uçnoktalara aynı veya benzer sorguların gönderildiği savunulmaktadır. Buradan yola çıkarak uçnoktalara atılan benzer sorguların taslağını sorgu eşleme yöntemi ile çıkarmak amaçlanmaktadır. Böylece elde edilen taslak sorgu çalıştırılıp ana sonuç önceden getirilmekte ve böylece kapsanan benzer sorguların cevabı arandığında, ana sorgu cevabı içerisinden elde edilebilmektedir. Gerçekleştirilen bu yöntemin katkısı DBPedia 3.6 SPARQL sorgu kayıtları tekrar çalıştırılarak değerlendirilmiştir.

Tüm bu bahsedilen SPARQL önbellekleme yaklaşımları, çeşitli katkılar sunmayı amaçlamaktadır ancak bazı eksik yönleri de mevcuttur. (Williams and Weaver, 2011)'ın yaklaşımı önbelleklemedeki sonuçların HTTP yanıt başlığına göre yönetilmesini amaçlar, fakat önbelleğin büyüklüğünü yönetmek için etkin bir mekanizma sunmaz. (Martin et al., 2010)'un yaklaşımında uygulama nesnelerinin önbelleklenmesi sağlanarak etkin bir yönetim gerçekleştirilir ancak birleştirilmiş SPARQL sorgularına uygun bir önbellekleme yaklaşımı tasarlanmamıştır. (Johannes Lorey, 2013)'nin taslak sonucunu önceden getirme yaklaşımı gerçekte büyük veri kümeleri üzerinde uygulanabilir bir yaklaşım değildir, çünkü taslak sorgunun genelliği arttığı zaman çok fazla veri getirme durumunda kalınmakta ve bu da sorgu sonucunun uzun süre beklenmesine yol açmaktadır.

Bu tez kapsamında gerçekleştirilen SPARQL önbellekleme yöntemi, sorgu sonuçlarının bellekte tekil üçlüler halinde tutulmasını amaçlar. Böylece hem tüm sorgu sonuçları bağlı verinin doğasına uygun olarak üçlüler şeklinde saklanır, hem de yeni sorgu eklendikçe bu üçlülerin önbellekte tek bir çizge halinde büyümesi sağlanarak bellek etkin bir şekilde yönetilmiş olur. Ehcache [5] aracı ile, önbellekteki sorguların

[4] http://staging.vakantieland.nl

[5] http://ehcache.org/

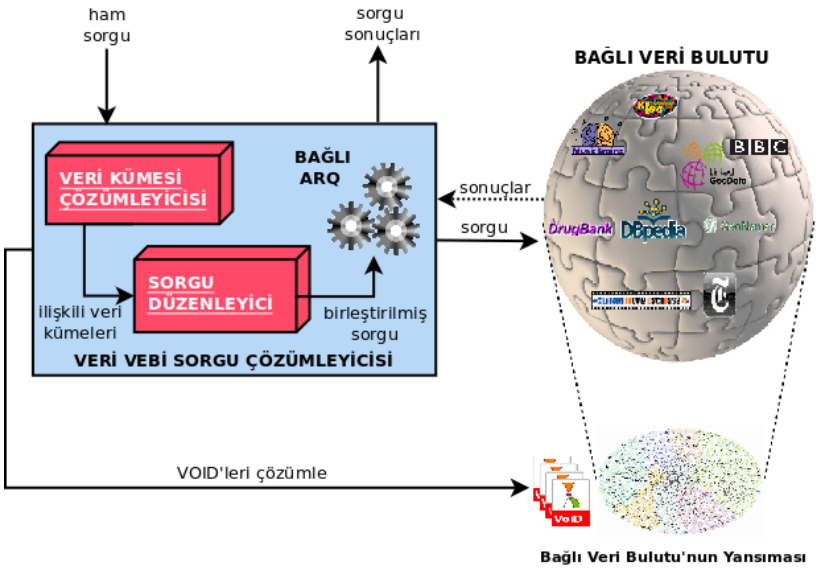
tahliyesi LRU (Least Recently Used), LFU (Least Frequently Used) veya FIFO (First In First Out) yaklaşımlarından birine göre, önbellekteki nesnelerin güncelliği de TTL (Time to Live) ve TTI (Time to Idle) değerleri ile, güvenli ve etkin bir şekilde yönetilmektedir. Gerçekleştirilen mimaride tek uçnokta üzerinde çalışan sorgulara ek olarak, SPARQL sorguları da desteklenmektedir. Önbellek mimarisi, hem Pareto Dağılımı ile genişletilen Berlin SPARQL Kıyaslama Kütüphanesi ile tekil SPARQL sorguları için, hem de FedBench Kıyaslama Kütüphanesi ile birleştirilmiş SPARQL sorguları için değerlendirilmiştir. Son olarak da Bellek Çizgesi, WoDQA'ya dahil edilerek, performansına olan katkısı ortaya konmuştur.

3 BAĞLI VERİ SORGULAMA MOTORU WoDQA'DA ENİYİLEŞTİRME GERÇEKLEŞTİRİMİ

Bu bölümde WoDQA'nın iç mimarisi ve gerçekleştirilen eniyileştirmeler ayrıntılı olarak anlatılmaktadır. WoDQA bağlı veri bulutu üzerinde sorgu çalıştırmayı amaçlamaktadır. Sorgulanmak istenen bağlı veri bulutu kurum-içi bağlı veri, kurumlar-arası bağlı veri, açık bağlı veri veya bunların bir birleşimi olabilmektedir. Bir sorguyu dağıtmak amacıyla, WoDQA sorguyu yalnızca ilişkili veri kümelerinde çalışan birleştirilmiş bir şekle dönüştürür. Bir sorgunun ilişkili veri kümelerini belirlemek için buluttaki veri kümelerini temsil eden VOID belgeleri kullanılır. VOID belgeleri bağlı veri bulutunun bir yansıması olarak düşünülebilir. Veri kümesi yayınlayıcıları veya kullanıcıları WoDQA' nın ihtiyaç duyduğu VOID belgelerini kendileri oluşturabilirler. Bunun yanında değişken bağlı veri bulutlarında, veri yayınlayıcıların kendi veri kümelerinin iyi tanımlanmış temsillerini yayınlamaları ve bunları güncel tutmaları daha uygun olacaktır. Böylece veri yayınlayıcıları kendi veri kümelerinin, sorgularda ilişkili veri kümesi olarak seçilebilmesini sağlamış olurlar. WoDQA bir sorguyu ilişkili veri kümeleri üzerinde çalıştırmak için gerekli 3 ana birimi içerir: Veri Kümesi Çözümleyicisi (Dataset Analyzer), Sorgu Düzenleyici (Query Reorganizer) ve Bağlı ARQ (Bound ARQ) (Bkz. Şekil 3.1)

Veri Kümesi Çözümleyicisi VOID belgelerini kullanarak ilişkili veri kümelerini belirlemek ve ilişkili olmayanları elemek ile sorumlu birimdir. Sorgulamada tam sonuca erişebilmek için veri kümesi yayınlayıcılarının bu belgelerin tam ve tutarlı olmasını sağladıklarını varsaymaktayız. Veri Kümesi Çözümleyicisi sorgudaki her bir üçlü deseni için veri kümesi ve bağ kümesi tanımlarını çözümler. Bu çözümleme sorgunun sonucunu etkilemeyen veri kümelerini eler. WoDQA veri kümesi çözümlemesini kural tabanlı olarak gerçekleştirir. Bölüm 3.2 ilişkili veri kümelerini bulan kuralları açıklamaktadır.

İkinci birim olan Sorgu Düzenleyici, Veri Kümesi Çözümleyicisi'nden gelen sonuçlara göre sorguyu yeniden oluşturur. Bu yeniden oluşturma süreci iki eniyileştirme adımını içerir. İlk adım aynı veri kümesinden sorgulanacak olan üçlü desenleri için harici grupları yaratma adımıdır. Diğer adım da harici gruplardaki üçlü desenlerini kendi içerisinde sıralama ve harici grupları kendi arasında sıralamadır. Son olarak,



Şekil 3.1. WoDQA iç mimarisi

SERVICE ifadelerini de içeren birleştirilmiş SPARQL sorguları oluşturulur[1]. Sorgu Düzenleyici ile ilgili ayrıntılar Bölüm 3.3 içerisinde verilmiştir.

Son modül, bir SPARQL sorgulama motoru olan Jena ARQ'yu FILTER tabanlı bağlı birleştirme gerçekleştirimi ile genişleten Bağlı ARQ'dur. Sorgu Düzenleyici ile oluşturulmuş SPARQL sorgularını çalıştırır. Bağlı ARQ sorgunun çalıştırılması sırasında atılan HTTP isteği sayısını azaltır ve sonuçlar daha hızlı bir şekilde elde edilebilir. Bağlı ARQ'nun iç işleyişi Bölüm 3.4'te tanıtılmaktadır. Bölüm 3.1'te WoDQA'nın birimlerini detaylandırmadan önce sonraki alt bölümlerde kullanmak üzere bazı ön tanımlar verilecektir.

3.1 Ön Tanımlar

WoDQA bağlı veri bulutları üzerinde çalıştığından, önce WoDQA'nın iç işleyişini tanımlayabilmek için bağlı veri bulutunun bir tanımı verilecektir. Özet olarak bağlı veri bulutu farklı veri kaynakları arasında tanımlı bağlar ile oluşturulmuş bir RDF çizgesidir. Temel olarak bir RDF çizgesi $(\mathcal{G})$, şekil olarak IRI (Internationalized Resource Identifier) lerin kümesinin (Duerst and Suignard, 2005) $\mathcal{I}$, boş düğümlerin kümesinin $\mathcal{B}$, literallerin kümesinin $\mathcal{L}$ ve tüm RDF terimlerinin $\mathcal{T} = \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$ olarak tanımlandığı, $\langle s, p, o \rangle$: $\mathcal{G} = \{\langle s, p, o \rangle \mid \langle s, p, o \rangle \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})\}$ şeklinde üçlülerin bir kümesi olarak temsil edilebilir. Bu doğrultuda bir bağlı veri bulutu, belirli bir veri uza-

[1] http://www.w3.org/TR/sparql11-federated-query/

yındaki IRI'ler, boş düğümler ve literallerden oluşan üçlülerin oluşturduğu tüm veriyi içeren bir $(\mathcal{G}_{cloud})$ çizgesi olarak ifade edilebilir.

Bir diğer bakış açısından, bir bağlı veri bulutu, birbiri ile bağlanmış veri kümelerinin bir ağı olarak düşünülebilir.Bağlı Açık Veri Bulutu da (Bizer et al., 2007) veri kümelerinin açık ve veri ölçekli bir ağı olarak tanımlanabilir. Bir veri kümesi (δ), RDF üçlülerinin anlamlı bir kümesidir ve bağlı veri bulutunun daha iri taneli bir şekilde görülmesini sağlar (Alexander et al., 2009).

Veri kümeleri, bilgiyi sadece tekil kaynaklar olarak yayımlama ve bu kaynakları bağlamadan farklı olarak, bilgiyi $\mathcal{G}_{cloud}$'un alt çizgeleri olarak paylaşmanın bir yoludur. Bu alt çizgeler, farklı veri kümelerindeki (Heath and Bizer, 2011) kaynakları birbirine bağlayan RDF üçlüleri aracılığıyla birbirine bağlıdır. Yayımlayıcılar kendi kaynaklarını yaratıp bunları ağ üzerindeki diğer veri kümeleri ile ilişkilendirir ve bilgi kullanıcıları da bu kaynakları kendi veri kümelerini yaratırken kullanırlar. Buna ilişkin, bir veri kümesini biçimlendirmek için, bir çizgedeki üçlülerin özneleri olan kaynak kümesini temsil eden $subj\,(\mathcal{G})$ kullanılır.

Bir veri kümesi bağlı veri bulutunun bir alt çizgesidir, $\delta_x \subset \mathcal{G}_{cloud}$, ve δ_x tarafından dahil edilen kaynaklar $Owner\,(\delta_x, r)$ ile ifade edilir ve şu şekilde tanımlanmıştır:

$\forall r, \delta_i\,(r \in subj\,(\delta_x) \rightarrow Owner\,(\delta_x, r))$.

Bir veri kümesinin içerdiği veriyi tanımlamak için VOID, iyi tanımlanmış özellikler içerir. Bir veri kümesinin VOID'ini bir kayıt(tuple) olarak $\langle \mathcal{L}^{space}, \mathcal{I}^{voc} \rangle \in \mathcal{L} \times \mathcal{I}$ tanımlayabiliriz. Bir veri kümesindeki tüm varlık(entity) IRI lerinin başladığı ön eklerin metin değerleri kümesi `void: uriSpace`, $\mathcal{L}^{space}$olarak adlandırılır. Diğer özellik $\mathcal{I}^{voc}$, veri kümesinde kullanılan sözlükler `void:vocabulary` olarak tanımlanır[2]. Bunların yanında VOID bazı istatistiksel özelliklerin tanımlanmasına da olanak verir, ancak WoDQA'ya bu tip istatistiksel özellikler dahil edilmemiş ve biçimlendirmede de bunlara yer verilmemiştir.

Nesnesi bir başka veri kümesinde özne olan üçlüler, veri kümesi bulutunun, birbirine bağlanmış veri kümelerinin bir çizgesi olmasını sağlar. Bu üçlülere bağ üçlüleri adı verilir ve burada iki tip bağ üçlüsü tanımlanacaktır. İlki düzenli bağ üçlüsü olarak tanımlanır ve

[2] $\mathcal{I}^{voc}$ kümesi anlamsal vebe genelleşmiş olan RDFS and OWL şemalarını içermez.

$s_{tp_{link}}, o_{tp_{link}} \in \mathcal{I}$, $Owner\left(\delta_x, s_{tp_{link}}\right)$, $Owner\left(\delta_y, o_{tp_{link}}\right)$ ve $\delta_x \neq \delta_y$ iken $tp_{link} = \langle s_{tp_{link}}, p_{tp_{link}}, o_{tp_{link}} \rangle$ olarak ifade edilebilir. Düzenli bağ üçlüleri bağlı veri bulutundaki farklı veri kümelerini birbirine bağlar. Diğer bağ üçlüsü tipi de sanal bağ üçlüsüdür. Bunu açıklamak için bağlı veri bulutunda yer alan veri kümelerinin kümesini Δ olarak temsil edelim. Sanal bağ üçlüsünün nesnesi bağlı veri bulutundaki hiçbir veri kümesi tarafından içerilmemektedir ve $s_{tp_{vl}}, o_{tp_{vl}} \in \mathcal{I}$, ve $\forall \delta_x \in \Delta\left(o_{tp_{vl}} \notin subj\left(\delta_x\right)\right)$ iken $tp_{vl} = \langle s_{tp_{vl}}, p_{tp_{vl}}, o_{tp_{vl}} \rangle$ olarak biçimlendirilebilir. Bu yöntem birbirine bağlı veri kümelerini ifade etmek için, veya sahibi bağlı veri bulutundan çıkarılmış veri kümelerinin bağlarının kopmadan devam edebilmesi için tasarlanmış mimari bir çözümdür.

VOID'in önemli bir katkısı da iki veri kümesi arasındaki bağ üçlülerini temsil eden bağ kümesi kavramıdır. Bağ kümesi kavramını $\left\langle \delta_\lambda^{from}, \delta_\lambda^{to}, p_\lambda^{link} \right\rangle \in \Delta \times \Delta \times \mathcal{I}$ kümesindeki bir kayıt(tuple) olarak λ şeklinde biçimlendirebiliriz. Bu tanımdan yola çıkarak, bağ üçlülerinin öznelerinin kümesi başvuran δ_λ^{from} olarak, bağ üçlülerinin nesnelerinin kümesi başvurulan δ_λ^{to} olarak, ve bağ kümesindeki tüm bağ üçlülerinin yüklemi de p_λ^{link} olarak tanımlanır. p_λ^{link}, `void:linkPredicate` özelliğine karşılık gelmektedir.

Sanal bağ üçlülerinin $\delta_\lambda^{to} \in \Delta^v$ olduğu durumda δ_λ^{to} sanal veri kümesi olmaktadır. Bir sanal veri kümesinin urispace dışında bir özelliği bulunmamakadır. Sanal veri kümeleri ilişkili olarak seçilmezler ancak veri kümesi çözümlemesinde sanal link durumları ile başa çıkabilmek için kullanılırlar.

WoDQA, ilişkili veri kümelerini seçmek için VOID üstverisinin hem veri kümesi hem de bağ kümesi tanımlarını kullanır. Çözümlenen sorgu seçilen veri kümeleri dahil edilerek birleştirilmiş sorguya dönüştürülür ve bu sorgu ilişkili veri kümeleri üzerinde işletilir. VOID ile üçlü desenleri arasındaki ilişkilerin çözümlenmesinin yanında, üçlü desenlerinin birbiri ile olan ilişkileri de dikkate alınır. Bunun için bir SPARQL sorgusunun biçimsel tanımını vermemiz gerekir. SPARQL sorgularının bir alt kümesini bir temel çizge deseni olarak düşünebiliriz (Buil et al., 2011). Bir temel çizge deseni üçlü desenlerinden oluşur, $bgp = \{tp_i | \langle s_{tp_i},\ p_{tp_i}, o_{tp_i} \rangle \in (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{L} \cup \mathcal{V})\}$. Bir üçlü deseni RDF üçlüsünden biraz farklıdır ve sonsuz $\mathcal{V}^3$ kümesinin elemanları olan en az bir, en çok üç değişken içerir. Bir sorguyu çalıştırırken sorgudaki değişkenler RDF terimleri ile değiştirilir. Böylece SPARQL'in anlamına göre, bir sorgu sonucu, sonuç eşlemelerinin bir kümesidir, $\{b | b : \mathcal{V} \rightarrow \mathcal{RN}\}$,

[3]Sorgularda boş düğümler göz ardı edilmiştir.

ve bir sonuç eşlemesi, b, değişkenlerden RDF terimlerine tanımlı parçalı bir fonksiyondur.

SPARQL notasyonunda kullanılan alan(domain) ve karşı alan (codomain) kavramları açıklanacak olursa, öncelikle X ve Y iki küme, Z ise X kümesinden Y kümesine bir fonksiyon $Z|Z : X \rightarrow Y$ olsun. Bu durumda Z fonksiyonunun alanı $dom(Z) = X$, karşı alanı ise $cod(Z) = Y$ şeklinde tanımlanır.

Son olarak algoritmalarda kullanılan kümeler üzerinde birleşim, kesişim ve fark işlemleri açıklanacaktır. İlk olarak $A = \left\{x_1, .., x_n\right\}$ ve $B = \left\{x_1, .., x_m\right\}$ şeklinde elemanları özdeş iki küme tanımlayalım ve $m < n$ olduğunu varsayalım. Bu durumda iki kümenin birleşimi $A \cup B = \left\{x_1, .., x_n\right\}$, iki kümenin kesişimi $A \cap B = \left\{x_1, .., x_m\right\}$, A kümesinin B kümesinden farkı $A \setminus B = \left\{x_{m+1}, .., x_n\right\}$ ve son olarak B kümesinin A kümesinden farkı $B \setminus A = \emptyset$ olacaktır.

3.2 Veri kümesi Çözümleyicisi

Bu bölümde WoDQA'nın özgün yanını oluşturan Veri kümesi Çözümleyicisi modülünden bahsedilecektir. Diğer sorgu birleştirme yaklaşımlarından farklı olarak, WoDQA sorgulanacak veri kümelerini belirlerken sorgu içerisindeki üçlü desenleri arasındaki ilişkileri ve veri kümeleri arasındaki bağları dikkate almaktadır. WoDQA'nın veri kümesi çözümlemesi sayesinde pek çok ilişkisiz veri kümesi elenirken ilişkili olan veri kümeleri belirlenmektedir. Veri kümesi çözümlemesi sonucunun çıktısı veri vebi üzerinde yayımlanmış olan tüm veri kümelerinin bir alt kümesidir ve sorgu sadece ilişkili veri kümelerini içeren bu alt küme üzerinde çalıştırılır.

Veri kümesi Çözümleyicisi, ilişkili veri kümelerinin seçilmesi için diğer bir deyişle sorgularla alakalı sonuç içerebilecek alt çizgelerin seçilmesi için VOID belgelerindeki veri kümesi tanımlamalarını ve bağ kümesi tanımlamalarını kullanır. VOID belgeleri içerisindeki veri kümesi ve bağ kümesi tanımlarınının çözümlenmesinin yanısıra sorgudaki üçlü desenlerinin kendi aralarındaki ilişkiler de dikkate alınır.

Bir sorgunun çözümlenmesinden sonra ilişkili veri kümesi bulunamadığı durumda, sorgunun tüm veri vebi üzerinde çalıştırılabilmesi için içerisindeki her üçlü deseninin veb üzerinde yayımlanmış tüm veri kümeleri üzerinde sorgulanması gerekir. Ancak bu, veb üzerindeki veri kümelerinin çokluğundan dolayı mümkün olmayacağı için WoDQA'nın çözümleme sürecinde ilişkisiz veri kümelerinin elenmesi amaçlan-

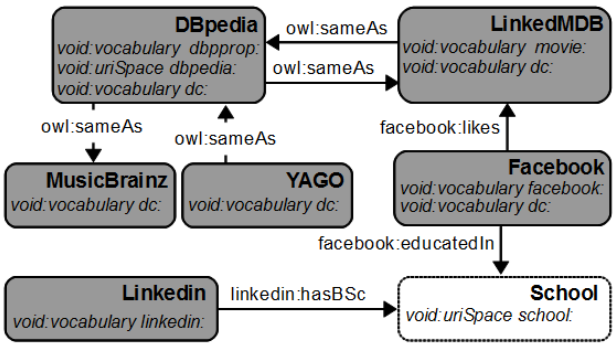
maktadır. Böylece oluşturulan birleştirilmiş sorgudaki alt sorgular sadece ilgili veri kümelerine dağıtılır ve sorgu çalıştırılır. İlişkisiz veri kümelerinin elenebilmesi için, ilişkili veri kümelerini ortaya çıkaran ilişkili veri kümesi keşif kuralları olarak adlandırılan bazı kurallar tanımlanmıştır. Bir üçlü deseni için ilişkili olan bir veri kümesi $\rho(\delta_x, tp_i)$ ilişkili veri kümesi belirtimi ile gösterilir ve burada δ_x veri kümesinin tp_i üçlü deseni için bir sorgulama sonucu içerebileceği anlamına gelmektedir. Veri kümesi keşif kuralları da bu ilişkili veri kümesi belirtimlerinin bulunmasını sağlarlar.

Bu noktada veri kümesi keşif kuralları sayesinde ilişkisiz veri kümelerinin nasıl elendiği açıklanacaktır. $\mathcal{Q}_{tp_i}$, tp_i ile ilişkili olan tüm veri kümelerinin tutulduğu kümeyi temsil etmektedir ve bu kümenin herhangi bir eleme yapılmadan önce veb üzerinde yayımlanan tüm veri kümelerini içerdiği varsayılır, yani ilk durumda $Q_{tp_i}^{init} \equiv \Delta$ olarak gösterilir. Her bir veri kümesi keşif kuralı Q_{tp_i} içerisindeki veri kümelerini dikkate alarak işletilir. Her bir kural uygulandıktan sonra ilişkili veri kümesi belirtimleri çıkarsanır. Çıkarsanan bu belirtimlerdeki veri kümeleri dışındaki veri kümeleri kesinlikle ilişkisizdirler ve Q_{tp_i} kümesinden uygulanan kural ile çıkarılarak elenmiş olurlar. Fakat her zaman bir kuralın uygulanması ile ilişkili veri kümesi çıkarsanmayabilir. Bu durumda ilişkili olan bir veri kümesi bulunmadığından Q_{tp_i} kümesindeki veri kümeleri için ilişkisiz olma durumu da çıkarsanmaz ve bir eleme yapılmaz. Bir kural uygulandıktan sonra Q_{tp_i}'nin ilişkisiz veri kümelerini eleme metoduyla güncellenerek ilişkisiz veri kümelerinden arındırılması şu şekilde biçimselleştirilmiştir:

Teorem 1 *Bir keşif kuralı uygulandıktan sonra en az bir tane ilişkili veri kümesi bulunduysa, bulunan bu ilişkili veri kümelerinden $\mathcal{Q}_{tp_i}$ kümesinde olanlar $\mathcal{Q}_{tp_i}^{new}$ kümesinin elemanıdır, $\mathcal{Q}_{tp_i}$ kümesindeki ilişkisiz veri kümeleri elenmiş olurlar. Ancak uygulanan keşif kuralı ile hiç bir ilişkili veri kümesi belirtimi bulanamıyorsa, $\mathcal{Q}_{tp_i}^{new}$ kümesi aynen $\mathcal{Q}_{tp_i}$ olarak kalır çünkü elenecek ilişkisiz veri kümesi de bulunmamış olur.*

$$\mathcal{Q}_{tp_i}^{new} = \left\{ \begin{array}{cc} \exists \delta_a \left(\rho\left(\delta_a, tp_i\right)\right); & \{\delta_x | \left(\delta_x \in \mathcal{Q}_{tp_i}\right) \wedge \rho(tp_i, \delta_x)\} \\ \exists^0 \delta_a \left(\rho\left(\delta_a, tp_i\right)\right); & \mathcal{Q}_{tp_i} \end{array} \right\}$$

Bundan sonraki bölümlerde veri kümesi keşif kuralları ayrıntılarıyla ele alınacaktır ve kullanımları bazı örnek sorgular üzerinde gösterilecektir. Şekil 3.2 veri kümesi keşif kurallarının bazı örnek sorgular üzerinde uygulanmaları için kullanılacak olan veri kümesi bulutunun VOID modellerini göstermektedir. Bu bulut yapay bir sos-



Şekil 3.2. Örnek bir bağllı veri bulutu

yal platform için gerekli veriyi temsil etmektedir. DBpedia, YAGO, MusicBrainz, ve LinkedMDB senaryoda yer alan bağlı açık veri bulutunun veri kümeleridir. Facebook ve Linkedin veri kümeleri sırasıyla Facebook ve Linkedin kullanıcılarının verisini içeren yapay RDF veri kümeleridir [4]. Okul veri kümesi Facebook ve Linkedin veri kümelerindeki kaynakların ortak kullandığı eğitim verisi sağlayan bir sanal veri kümesidir. Şekilde bağ yüklemleri, kutular ile temsil edilen veri kümeleri arasındaki oklarla temsil edilmiştir. Şekil 3.2'de daha fazla bağ kümesi olabilecekken, kurallarımızı açıklamak için sadece birkaç tanesi dikkate alınmıştır.

3.2.1 İlişkili veri kümesi keşif kuralları

Bu bölümde, ilişkili veri kümelerinin etkin olarak belirlenmesinde kullanılan keşif kurallarından söz edilecektir. Her bir keşif kuralı, veri kümelerini farklı bakış açılarından çözümleyip üçlü desenlerinin bu veri kümelerinden sorgulanmak üzere ilişkili olup olmadıklarına karar vermekte ve ilişkili veri kümesi belirtimlerini (ρ) çıkarsamaktadır. Keşif kurallarının ele alındığı bakış açıları üç grupta toplanabilir. İlk bakış açısı olan IRI Tabanlı Çözümlemede, üçlü desenleri içerisindeki IRI'ler çözümlenir. Sorgulardaki özne ve nesne konumundaki IRI'lerin isim uzayları ve üçlü desenlerinde IRI olarak bulunan yüklemler ile VOID belgelerindeki void:uriSpace ve void:vocabulary tanımlamaları çözümlenip veri kümelerinin üçlü desenleri için ilişkili olup olmadıklarına karar verilir. İkinci bakış açısında birbirine bağlı kaynaklar dikkate alınmaktadır. Bağ Çözümlemesi denilen bu bakış açısında birbirine bağlı iki kaynağın aynı veri kümesinde ya da farklı veri kümelerinde olmalarına göre farklı değerlendirmeler yapılır.

[4]Facebook ve Linkedin veri kümeleri örneğe yapay olarak eklendiğinden bu önekler temsili bir şekilde kullanılmış ve Tablo 4.5'te yer almamıştır.

Son bakış açısı Paylaşılan Değişken Çözümlemesi'dir. Üçlü desenleri aynı değişkenleri içerebilirler ve bu durum birbirlerinin ilişkili veri kümelerini etkileyebilir. Bu son çözümleme bakış açısı üçlü desenlerinin ortak değişkenlerine bakarak ilişkisiz veri kümelerini elemeyi amaçlar. Tüm bakış açıları için ilişkili veri kümesi keşif kuralları aşağıda anlatılmaktadır.

Şimdi tanıtılacak olan ilk iki keşif kural IRI Tabanlı Çözümleme bakış açısı altında sınıflandırılmaktadır. İkisi de VOID belgelerindeki void:vocabulary $(\mathcal{I}^{voc})$ ile belirtilen veri kümesi tarafından kullanılan sözlükleri dikkate alır. İlk keşif kuralı, üçlü desenlerindeki IRI olarak bulunan yüklemlerin VOID belgelerindeki tanımlı sözlüklere $(\mathcal{I}^{voc})$ ait bir özellik olup olmadıklarını kontrol eder. Sözlük Eşleşmesi Kuralı'nı biçimsel olarak tanımlayabilmek için, bir kaynağın $(r \in \mathcal{I})$, bir veri kümesinin VOID belgesinde tanımlı herhangi bir sözlüğe $(\mathcal{I}^{voc}_{\delta_x})$ ait olup olmadığını gösteren $has(\mathcal{I}^{voc}_{\delta_x}, r)$ fonksiyonu kullanılacktır. Böylece veri kümesinde kullanılan sözlük tanımlarına göre bir kaynağın hangi veri kümeleri ile ilişkili olup olmadığı Tanım 1'deki gibi gösterilir.

Tanım 1 *Eğer bir kaynak bir veri kümesinin tanımlı sözlüklerinden herhangi birine aitse o kaynak veri kümesinin sözlükleriyle eşleşiyordur.*

$\forall \delta_x(has(\mathcal{I}^{voc}_{\delta_x}, r) \rightarrow VocMatch(\delta_x, r))$, $r \in \mathcal{I}$ *iken*

"`?s dbpprop:name 'Nikola Tesla`"', gibi bir üçlü deseni için, yüklem konumunda bulunan dbpprop:name RDFS özelliği, üçlü deseninin dbpprop sözlüğünü kullanan bir veri kümesinden sorgulanmasını zorunlu kılmıştır. Çünkü bu sözlükte kullanılan bir kaynak, üçlü deseni içerisinde yüklem olarak kullanılmıştır ve sadece dbpprop sözlüğünü kullanan veri kümelerinin bu üçlü deseni için bazı üçlüler içerme olasılığı vardır. Bu nedenle dbpprop sözlüğünü kullanmayan veri kümeleri elenmelidir. Kural 1 , yani Sözlük Eşleşmesi Kuralı ile, keşfedilmiş ilişkili veri kümeleri dışındaki veri kümeleri elenir.

Kural 1 *Eğer bir üçlü deseninin yüklemi bir veri kümesinin kullandığı sözlüklerden herhangi birisi ile eşleşiyorsa, veri kümesi üçlü deseni için ilişkilidir.*

$$\forall tp_i, \delta_x \left(VocMatch \left(\delta_x, p_{tp_i} \right) \rightarrow \rho \left(\delta_x, tp_i \right) \right)$$

Şekil 3.2'ye göre DBpedia veri kümesinin dbpprop sözlüğünü kullandığı görülmektedir ve DBpedia'nın bu üçlü deseni için ilişkili olduğu sonucu çıkarılır. Veri

vebi üzerinde herhangi bir sözlüğü kullanan çok fazla veri kümesi bulunabilir. Onların içerisinde bulunan ilişkisiz olan veri kümelerini de elemek için ileride anlatılacak olan keşif kuralları kullanılacaktır.

IRI Tabanlı Çözümleme altındaki ikinci kural RDF Tip Eşleşmesi kuralıdır. Bu kuralda içerisinde rdf:type yüklemini içeren üçlü desenlerinin nesnelerindeki IRI'lerin hangi sözlükler ile eşleştikleri incelenir. Bunun için `?producer rdf:type lmdbvoc:producer`, gibi bir üçlü deseni örneğini ele alacağız. Bu örnekteki üçlü deseninin nesnesi linkedMDB sözlüğünü kullanan veri kümeleri dışındaki veri küme-lerinin elenmesini sağlar. Şekil 3.2'deki VOID modellerine göre LinkedMDB veri kümesi bu sözlüğü kullanmaktadır ve ilişkili bir veri kümesidir. Kural 2, RDF tip eşlemesi kuralını göstermektedir.

Kural 2 *Eğer bir üçlü deseninin yüklemi rdf:type ise ve o üçlü deseninin nesnesi bir veri kümesinin tanımlı sözlüklerinden herhangi biri ile eşleşiyorsa, o veri kümesi o üçlü deseni için ilişkilidir.*

$\forall tp_i, \delta_x(VocMatch(\delta_x, o_{tp_i}) \wedge (p_{tp_i} = \texttt{rdf:type}) \rightarrow \rho(\delta_x, tp_i))$ ***iken***

İlişkili veri kümelerini keşfetmede kullanılan bir diğer bakış açısı Bağ Çözümlemesi'dir. Bir üçlünün öznesi ve nesnesi bir IRI olduğunda üçlü iki kaynağı birbirine bağlıyor demektir. Nesne konumundaki kaynak özne ile aynı veri kümesine ait bir kaynak olabileceği gibi farklı veri kümesine ait bir kaynak da olabilir. Bu yüzden bu gibi üçlü desenlerini iki aşamada çözümlemek gerekir. İlk aşamada İçsel Bağ Çözümlemesi gerçekleştirilir. Bu çözümleme, üçlülerin aynı veri kümesi içerisindeki kaynakları birbirine bağladığını varsayarak ilişkili veri kümelerini keşfeder. Bu çözümleme ile bulunan ilişkili veri kümeleri içsel ilişkili veri kümesi olarak adlandırılır ve $\rho^{int}(\delta_x, tp_i)$ belirtimi ile gösterilir. Diğer aşamada Dışsal Bağ Çözümlemesi yapılır. Üçlülerin bağladığı kaynakların farklı veri kümesinde oldukları varsayılarak çözümleme yapılır. Bu durumda VOID belgelerindeki bağ kümesi tanımlamaları dikkate alınır. Bu çözümleme ile bulunan ilişkili veri kümeleri dışsal ilişkili veri kümesi olarak adlandırılır ve $\rho^{ext}(\delta_x, tp_i)$ belirtimi ile gösterilir. Bir üçlü deseni için Bağ Çözümlemesi bakış açısı uygulanırken hem İçsel hem de Dışsal Bağ Çözümlemesi yapılır ve üretilen içsel ve dışsal ilişkili veri kümeleri üçlü deseni için ilişkili veri kümesi olarak kabul edilirler. Kural 3'te bir üçlü deseni için ilişkili veri kümelerinin, o üçlü için keşfedilen içsel ve dışsal ilişkili veri kümelerinin birleşimi olduğu ortaya konmaktadır.

Kural 3 *Bir üçlü deseni için üretilen içsel ve dışsal ilişkili veri kümelerinin tümü o üçlü deseni için ilişkili veri kümeleridir.*

$$\forall \delta_x, tp_i \left(\rho^{int}\left(\delta_x, tp_i\right) \vee \rho^{ext}\left(\delta_x, tp_i\right) \rightarrow \rho\left(\delta_x, tp_i\right)\right)$$

Kural 3 ile bulunan ilişkili veri kümeleri kullanılarak üçlü deseninin sorgulanacak veri kümeleri (Q_{tp_i}) içerisindeki ilişkisiz veri kümeleri elenir. İçsel ve Dışsal ilişkili veri kümeleri, üçlü desenleri için ilişkili veri kümelerinin çıkarsanması için ara sonuçlardır.

Bağ Çözümlemesi bakış açısının kullanıldığı ilk iki kural IRI'ye Bağlanma Kuralları olarak adlandırılır. Bunun için bir üçlüde özne ya da nesne konumunda olan bir kaynağın bir veri kümesine ait olma durumunu biçimsel olarak tanımlamamız gerekmektedir. Eğer bir kaynağın IRI'si bir veri kümesi için tanımlanan isim uzayı özelliklerinden $(\mathcal{L}_{\delta_x}^{space})$ herhangi biri ile başlıyorsa, veri kümesinin kaynağın sahibi olduğu çıkarsaması yapılabilir. Bu durum Tanım 2'de gösterilmiştir. `?film owl:sameAs dbpedia:A_Fistful_of_Dollars`, üçlü deseni örneğine göre `?film` değişkenine karşılık gelebilecek kaynaklar `dbpedia:A_Fistful_of_Dollars` kaynağına `owl:sameAs` yüklemi ile bağlı olan kaynaklardır ve bu kaynakların sahibi olan veri kümeleri de ilişkili veri kümeleridir. Tanım 2'deki bir kaynağın $(r \in \mathcal{I})$ bir veri kümesi (δ_x) tarafından içerilmesi durumunu, VOID tanımındaki URI uzayı $(\mathcal{L}^{space})$ özelliklerini kullanarak ve r 'nin $\mathcal{L}_{\delta_x}^{space}$'teki URI uzaylarından biri ile başladığını belirten $startsWith\left(r, \mathcal{L}_{\delta_x}^{space}\right)$ fonksiyonu ile biçimselleştirilmiştir.

Tanım 2 $\forall \delta_x(startsWith(r, \mathcal{L}_{\delta_x}^{space}) \rightarrow Owner(\delta_x, r))$

Kural 4 , İçsel Bağ Çözümlemesi bakış açısıyla çözümlemeleri gerçekleştiren IRI'ye Bağlanma İçsel Kuralı'nı göstermektedir. Yukarıdaki örnek sorguya göre ?film değişkenine karşılık gelen kaynaklar dbpedia:A_Fistful_of_Dollars kaynağı ile aynı veri kümesinde olmalıdır. Bu nedenle bu üçlü deseni, dbpedia:A_Fistful_of_Dollars kaynağının da sahibi olan DBpedia veri kümesinden sorgulanmalıdır.

Kural 4 *Bir üçlü deseninde nesne konumunda bulunan IRI'nin sahibi olan veri kümesi, o üçlü deseni için içsel ilişkili veri kümesidir.*

$$\forall tp_i, \delta_x \left(\left(\delta_x \in \mathcal{Q}_{tp_i}\right) \wedge Owner\left(\delta_x, o_{tp_i}\right) \rightarrow \rho^{int}\left(\delta_x, tp_i\right)\right)$$

where $o_{tp_i} \in \mathcal{I}, s_{tp_i} \in \mathcal{V}$

Diğer taraftan Dışsal Bağ Çözümlemesi bakış açısına göre uygun üçlüler, nesne IRI'sinin sahibi olan veri kümesine bağlanan veri kümelerinde bulunurlar. Örneğe göre ?film değişkenine karşılık gelen kaynaklar, nesneleri DBpedia'da tanımlı olan bağ üçlülerini içeren DBpedia'dan farklı veri kümelerinde bulunabilirler. Bu çözümlemeyi yapabilmek için VOID belgelerindeki bağ kümesi tanımlamaları kullanılır. IRI'ye Bağlanma çözümlemesini dışsal açıdan uygulayabilmemiz için üçlü deseni içerisindeki yüklemin bir IRI ve aynı zamanda bir bağ yüklem olması gerekmektedir. Üçlü desenleri için hangi bağ kümesi tanımlamalarının kullanılacağını Tanım 3'deki $Compatible$ bildirimi ile biçimsel olarak gösterilmektedir.

Tanım 3 *Eğer bir üçlünün o anki ilişkili veri kümesi kümesi* $(\mathcal{Q}_{tp_i})$ *bağ kümesi tanımlamasında yer alan kaynak veri kümesini içeriyorsa ve bağ kümesi tanımlamasında yer alan bağ yüklem üçlü desenindeki yüklem ile aynıysa bağ kümesi üçlü deseni için uygundur.* $\forall \lambda_m, tp_i((\delta^{from}_{\lambda_m} \in \mathcal{Q}_{tp_i}) \wedge (p^{link}_{\lambda_m} = p_{tp_i}) \rightarrow Compatible(\lambda_m, tp_i))$

```
?film owl:sameAs dbpedia:A_Fistful_of_Dollars
```

üçlü deseni örneğine ve Şekil 3.2'deki VOID modeli örneğine göre dbpedia:A_Fistful_of_Dollars kaynağına sahip olan DBpedia veri kümesine owl:sameAs bağ yüklemi ile sadece LinkedMDB ve NYtimes veri kümeleri bağlıdır. Buna göre Kural 5'teki IRI'ye Bağlanma Dışsal Kuralı bu iki veri kümesini dışsal ilişkili veri kümeleri olarak çıkarsamaktadır.

Kural 5 *Bir üçlü deseni ile uyumlu bir bağ kümesi tanımı varsa ve bu bağ kümesi tanımının hedef veri kümesi üçlü desenindeki nesne olan IRI'nin sahibi ise, o bağ kümesi tanımının kaynak veri kümesi, üçlü deseni için dışsal ilişkili veri kümesidir.*

$\forall tp_i, \lambda_m, \delta_x(Compatible(\lambda_m, tp_i) \wedge Owner(\delta_x, o_{tp_i}) \wedge (\delta_x = \delta^{to}_{\lambda_m}) \rightarrow \rho^{ext}(\delta^{from}_{\lambda_m}, tp_i))$, $o_{tp_i} \in \mathcal{I}, s_{tp_i} \in \mathcal{V}$ *iken.*

Kural 5, $Owner$ ifadesi URI uzayı ile karar verdiğinden hem düzenli hem de sanal bağları dikkate almış olur. İçsel ve dışsal ilişkili veri kümeleri Kural 4 ve Kural 5 ile bulunduktan sonra üçlü deseni için ilişkili veri kümesini oluşturmak amacıyla Kural 3 ile birleştirilmelidirler. Böylece, örnek üçlü deseni için, IRI'ye Bağlanma Kuralları'ne göre ilişkili veri kümeleri DBpedia, LinkedMDB ve YAGO veri kümeleridir.

Bağ Çözümlemesi bakış açısı altındaki diğer bir çift kural ise IRI'nin Bağları Kuralları olarak adlandırılır ve içsel ve dışsal açıdan ilişkili veri kümelerini bulmaya

yöneliktir. Bu kurallar öznesi bir IRI ve nesnesi bir değişken olan bir üçlü desenini çözümlemek için kullanılır. Kural 6, IRI'nin Bağları İçsel Kuralı'dır ve aynı veri kümesinde kaynakları birbirine bağlayan üçlüleri bulmayı hedefler. Bu kurala göre üçlü desenindeki öznenin sahibi olan veri kümesi, o üçlü deseni için içsel ilişkili veri kümesidir.

Kural 6 *Bir üçlü deseninin öznesinin sahibi olan veri kümesi, üçlü deseni için içsel ilişkili veri kümesidir.*

$\forall tp_i, \delta_x\ ((\delta_x \in \mathcal{Q}_{tp_i}) \wedge Owner(\delta_x, s_{tp_i}) \rightarrow \rho^{int}(\delta_x, tp_i)), o_{tp_i} \in \mathcal{V}, s_{tp_i} \in \mathcal{I}$ *iken.*

`dbpedia:Ennio_Morricone owl: sameAs ?person` örneğini ele alalım. Üçlü deseninin öznesinin isim uzayı, DBpedia VOID belgesinde tanımlı olan bir isim uzayı özelliğidir. Bu yüzden DBpedia, bu üçlü deseni örneği için içsel ilişkili veri kümesidir.

Kural 7 IRI'nin bağlanması dışsal keşif kuralıdır ve farklı veri kümelerini bağlayan kaynakları bulmaktadır. Buna göre eğer sahip olan veri kümesini başvuran olarak içeren ve üçlü deseni ile uyumlu bir bağ kümesi tanımı varsa bu durumda üçlü desenin öznesinin sahibi olan veri kümesi dışsal ilişkilidir.

Kural 7 *Bir üçlü deseni ile uyumlu bir bağ kümesi tanımı varsa ve bu bağ kümesi tanımının hedef veri kümesi üçlü desenindeki nesne olan IRI'nin sahibi ise o bağ kümesi tanımının kaynak veri kümesi üçlü deseni için dışsal ilişkili veri kümesidir.*

$\forall tp_i, \lambda_m, \delta_x(Compatible(\lambda_m, tp_i) \wedge Owner(\delta_x, s_{tp_i}) \wedge (\delta_x = \delta_{\lambda_m}^{from}) \rightarrow \rho^{ext}(\delta_x, tp_i))$
$, o_{tp_i} \in \mathcal{V}, s_{tp_i} \in \mathcal{I}$ *iken*

Örneğe göre, DBpedia `dbpedia:Ennio_Morricone`'nin sahibi olan veri kümesidir ve bağ yüklemi `owl:sameAs` olan ve başvuran veri kümesi DBpedia olan bir bağ kümesi tanımı vardır.

Yinelemeli Çözümleme aşamasındaki tüm kurallar Paylaşılan Değişken Çözümlemesi bakış açısı altındadır. Ayrıca bu bakış açısı altındaki kurallar uygulanırken IRI Tabanlı Çözümleme'de ve Bağ Çözümlemesi'nde belirtilen yöntemler de kullanılmaktadır. Paylaşılan Değişken Çözümlemesi altındaki ilk iki kural Zincirleme Üçlü Desenleri Kuralları olarak adlandırılır. Bu kurallar ilişkili veri kümelerini keşfedebilmek için birinin nesnesi diğerinin öznesi ile aynı değişkeni içeren iki üçlü desenini

birlikte ele alır.

```
?s owl:sameAs ?film.
?film lmdbvoc:producer_name 'Sergio Leone'.
```

üçlü desenleri ?film değişkenini hem özne hem de nesne olarak içerdikleri için birlikte ele alınabilirler. İçsel Bağ Çözümlemesi ile bakıldığında ?film değişkenine karşlık gelen kaynakların bulunduğu veri kümesi aynı zamanda ?s değişkenine kaşılık gelen kaynakların da bulunduğu veri kümesidir. Yani örneğe göre üçlü desenleri ?film değişkenine karşılık gelen kaynakların bulunduğu veri kümesinden sorgulanmalıdır. Bu yaklaşıma göre üçlü desenlerinin içsel ilişkili veri kümelerini bulabilmek için Zincirleme Üçlü Desenleri İçsel Kuralı, Kural 8 tanımlanmıştır.

Kural 8 *Eğer bir üçlü deseninin nesnesi başka bir üçlü deseninin öznesi ile aynıysa bu üçlü desenlerinin sorgulanacakları veri kümelerindeki ortak elemanların kesişimi, bu üçlü desenleri için içsel ilişkili veri kümeleridir.*

$\forall \delta_x, tp_i, tp_j((o_{tp_i} = s_{tp_j}) \wedge (\delta_x \in \mathcal{Q}_{tp_i}) \wedge (\delta_x \in \mathcal{Q}_{tp_j}) \rightarrow \rho^{int}(\delta_x, tp_i) \wedge \rho^{int}(\delta_x, tp_j))$, $o_{tp_i}, s_{tp_i} \in \mathcal{V}$ *iken.*

Zincirleme Üçlü Deseni Çözümlemesini işletmek için kuralların işletim sırası önemlidir. Bu durumu Zincirleme Üçlü Desenleri sorgusu ile örneklemek için, IRI-tabanlı çözümlemenin önce uygulandığını düşünelim ve LinkedMDB VOID'i, `linkedMDB` yi sözlük değeri olarak ikinci üçlü deseni için dahil eder. Daha sonra bu kural ilk üçlü deseni için içsel ilişkili veri kümesini LinkedMDB olarak belirleyebilir. Bu örnekten açıkça görüldüğü üzere, Paylaşılan Değişken Çözümlemesi IRI-tabanlı Çözümleme kurallarının daha fazla veri kümesi elemek için çalıştırılmasının ardından işletilmelidir. Alt bölüm 3.2.2'te bu kurallar için işletim sırasının belirlendiği çözümleme süreci anlatılmaktadır.

Sözlük Eşleşmesi Kuralı uygulandıktan sonra ikinci üçlü deseni için ilişkisiz olan veri kümeleri Şekil 3.2'de LinkedMDB veri kümesinin linkedMDB sözlüğünü kullanmasından dolayı elenir ve sonuçta $Q_{tp_2} = \{\delta_{LinkedMDB}\}$ olur ve birinci üçlü desenininki de $Q_{tp_1} \equiv \Delta$ 'dir. Daha sonra Zincirleme Üçlü Desenleri İçsel Kuralı şu belirtimleri çıkarsar: $\rho^{int}(\delta_{LinkedMDB}, tp_1)$, $\rho^{int}(\delta_{LinkedMDB}, tp_2)$. Yani birinci üçlü deseni için ilişkisiz veri kümeleri de elendikten sonra içsel ilişkili veri kümeleri sadece LinkedMDB olur.

Diğer yandan aynı örneğe Dışsal Bağ Çözümlemesi ile yaklaştığımızda ise üçlü desenleri incelenirken bağ kümesi tanımlamaları kullanılır. İçsel Bağ Çözümlemesi ile üçlü desenlerinde bağ yüklem olmasına bakılmazken, burada paylaşılan değişkeni nesne olarak içeren üçlü deseninin bağ yüklem içermesi beklenir.

Kural 9 *Eğer bir üçlü deseninin* (tp_i) *nesnesi bir diğer üçlü deseninin* (tp_j) *öznesi ile aynıysa ve* (tp_i) *üçlü deseni ile uyumlu bir bağ kümesi tanımı var ise ve bağ kümesi tanımının hedef veri kümesi* (tp_j) *üçlü deseninin o anki sorgulanacak ilişkili veri kümelerinden biri ise bağ kümesi tanımının kaynak veri kümesi* (tp_i) *üçlü deseni için, hedef veri kümesi de* (tp_j) *üçlü deseni için dışsal ilişkili veri kümesidir.*

$\forall \lambda_m, tp_i, tp_j((o_{tp_i} = s_{tp_j}) \wedge Compatible(\lambda_m, tp_i) \wedge (\delta^{to}_{\lambda_m} \in \mathcal{Q}_{tp_j}) \rightarrow \rho^{ext}(\delta^{from}_{\lambda_m}, tp_i) \wedge \rho^{ext}(\delta^{to}_{\lambda_m}, tp_j))$, $o_{tp_i} \in \mathcal{V}$ *iken.*

Zincirleme Üçlü Desenleri Dışsal Kuralı'nın örnek sorguya uygulanmadan önce üçlü desenleri için sorgulanacak ilişkili veri kümelerinin $Q_{tp_1} \equiv \Delta$ ve $Q_{tp_2} = \{\delta_{LinkedMDB}\}$ olduğunu varsayalım. Kural 9'a göre birinci üçlü deseni için dışsal ilişkili veri kümesi sadece DBpedia'dır. Çünkü ?film değişkenine karşılık gelen kaynaklar sadece LinkedMDB'dedir ve LinkedMDB veri kümesine owl:sameAs ile bağlı olan tek veri kümesi de DBpedia'dır. Sonuçta sorgulanacak dışsal ilişkili veri kümeleri $\rho^{ext}(\delta_{DBpedia}, tp_1)$ ve $\rho^{ext}(\delta_{LinkedMDB}, tp_2)$ olur. Zincirleme Üçlü Desenleri İçsel ve Dışsal Kuralları uygulanıp içsel ve dışsal ilişkili veri kümeleri çıkarsandıktan sonra Kural 3 ile birleştirilirler ve ilişkili veri kümeleri oluşturulur.

Zincirleme Üçlü Desenleri Kuralları'na benzer şekilde Paylaşılan Değişken Çözümlemesi bakış açısı altında incelenen diğer kural çifti ise Nesne Paylaşan Üçlü Desenleri Kuralları'dır. Bu kurallar aynı nesne değişkenini paylaşan üçlü desenlerini ele alıp bu üçlü desenleri için ilişkili veri kümelerini bulmaya çalışır. Bu kuralların açıklanması için aşağıdaki sorgu örneği kullanılacaktır:

```
?person facebook:likes ?movie .
?film owl:sameAs ?movie .
```

Örnekteki üçlü desenlerini İçsel Bağ Çözümlemesi ile ele aldığımızda ?person ve ?film değişkenlerine karşılık gelecek olan kaynaklar ?movie değişkenine karşılık gelen kaynakların bulunduğu veri kümesi ile aynı veri kümesinde olmalıdır. Dolayısıyla iki üçlü deseninin de içsel ilişkili veri kümeleri aynı olmalıdır. Üçlü desenlerinin o anki sorgulanacakları ilişkili veri kümelerinin $\mathcal{Q}_{tp_1} = \{\delta_{Facebook}\}$ ve $\mathcal{Q}_{tp_2} \equiv \Delta$ olduğunu varsayalım. Kural 10'da biçimselleştirilen Nesne Paylaşan Üçlü Desenleri İçsel

Kuralı'nın uygulanması ile iki üçlü deseni için de içsel ilişkili veri kümelerinin sadece $\delta_{Facebook}$ olduğu çıkarsanır. Çünkü sadece Facebook veri kümesi her iki üçlü deseni için de ortak çözümler içerir.

Kural 10 *Eğer bir üçlü deseninin nesnesi ile bir diğer üçlü deseninin nesnesi aynı ise, sorgulanacakları ilişkili veri kümeleri kümelerindeki ortak elemanlar her ikisi için de içsel ilişkili veri kümeleridir.* $\forall \delta_x, tp_i, tp_j((o_{tp_i} = o_{tp_j}) \wedge (\delta_x \in \mathcal{Q}_{tp_i}) \wedge (\delta_x \in \mathcal{Q}_{tp_j}) \rightarrow \rho^{int}(\delta_x, tp_i) \wedge \rho^{int}(\delta_x, tp_j))$, $o_{tp_i} \in \mathcal{V}$ *iken.*

Aynı sorgu örneğini Dışsal Bağ Çözümlemesi ile ele alabilmek için bağ kümesi tanımlamaları kullanılır. Bu bakış açısına göre üçlü desenlerinin en az birisinde bağ yüklem olmalıdır. Her iki üçlü deseninin de bağ yüklem içererek ortak bir nesne değişkeni içerdiği durum ele alındığında üçlü desenlerinin öznelerine karşılık gelen kaynaklar paylaşılan nesne değişkenine karşılık gelen kaynağın bulunduğu veri kümesinden farklı bir veri kümesinde bulunmalıdır. Sadece bir üçlü deseninin bağ yüklem içererek ortak bir nesne değişkeni içerdiği durumda bağ yüklem içeren üçlü deseni öznesine karşılık gelen kaynaklar paylaşılan nesneye karşılık gelen kaynaktan farklı bir veri kümesinde, diğer üçlü deseninin öznesine karşılık gelen kaynaklar ise paylaşılan nesneye karşılık gelen kaynaklar ile aynı veri kümesinde bulunmalıdır. Kural 11 ile biçimselleştirilen Nesne Paylaşan Üçlü Desenleri Dışsal Kuralları aynı nesneyi paylaşan üçlü desenleri için dışsal ilişkili veri kümelerinin bulunmasını sağlamaktadır. Bu kural en az bir üçlü deseninin bağ kümesi tanımlamalarına uymasını gerektirmektedir.

Kural 11 *Eğer iki üçlü deseni aynı değişkeni nesne olarak bulunduruyorsa ve her ikisi için de hedef veri kümeleri aynı olan ve üçlü desenleri için uygun olan bağ kümesi tanımlamaları varsa üçlü desenlerinin kendilerine uygun olan bağ kümesi tanımlamalarının kaynak veri kümeleri üçlü desenleri için dışsal ilişkili veri kümeleridir. Bu bağ kümelerinin hedef veri kümeleri sanal veri kümeleriyse uri uzayı eşlemesi dikkate alınır.* $\forall \lambda_m, \lambda_n, tp_i, tp_j((o_{tp_i} = o_{tp_j}) \wedge Compatible(\lambda_m, tp_i) \wedge Compatible(\lambda_n, tp_j) \wedge ((\delta^{to}_{\lambda_m} = \delta^{to}_{\lambda_n}) \vee ((\delta^{to}_{\lambda_m} \in \triangle^v) \wedge (\delta^{to}_{\lambda_m} \in \triangle^v) \wedge (\mathcal{L}^{space}_{\delta^{to}_{\lambda_m}} = \mathcal{L}^{space}_{\delta^{to}_{\lambda_n}}))) \rightarrow \rho^{ext}(\delta^{from}_{\lambda_m}, tp_i) \wedge \rho^{ext}(\delta^{from}_{\lambda_n}, tp_j))$, $o_{tp_i} \in \mathcal{V}$ *iken.*

Nesne Paylaşan Üçlü Desenleri Dışsal Kuralını açıklayalım. Her iki üçlü desenindeki yüklemlerin bağ yüklem olduğunu ve örnek üçlü desenlerine hiçbir çözümleme uygulanmadığını varsayalım. Buna göre üçlü desenleri için sorgulanacak veri kümeleri

kümeleri $\mathcal{Q}_{tp_1} \equiv \Delta$ ve $\mathcal{Q}_{tp_2} \equiv \Delta$ şeklindedir. Şekil 3.2'ye göre $\delta_{DBpedia}$, δ_{YAGO}, ve $\delta_{LinkedMDB}$ veri kümeleri `owl:sameAs` bağ yüklemi içermektedir. Sadece $\delta_{Facebook}$ veri kümesi `facebook:likes` bağ yüklemi içeren bağ üçlülerini içermektedir ve bu bağ üçlülerinin nesneleri de $\delta_{LinkedMDB}$ veri kümesinde bulunmaktadır. Buna göre birinci üçlü deseni için dışsal ilişkili veri kümeleri $Q^{new}_{tp_1} = \{\delta_{Facebook}\}$ şeklindedir. Bu durumda, tp_2 sadece to $\delta_{LinkedMDB}$ veri kümesine `owl:sameAs`, ile bağlı olan $\delta_{DBpedia}$'dan sorgulanmaktadır ve $Q^{new}_{tp_2} = \{\delta_{DBpedia}\}$ ilişkili olur. Çünkü ?film değişkeni ile eşleşen ve diğer veri kümelerinde tanımlı olan kaynaklar birinci üçlü deseni için bulunan ?movie değişkenine karşılık gelen kaynaklarla eşleşmeyecektir. Diğer bağ çözümlemesi yöntemlerinde yapıldığı gibi, Kural 10 ve Kural 11 den çıkarılan içsel ve dışsal ilişkili veri kümeleri birleştirilir.

Şimdi, Kural 11' in geliştirilmesiyle dahil edilmiş sanal veri kümesi tanımlarının kullanıldığı yeni bir örnek tanımlayacağız:

```
?facebookPerson facebook:educatedIn ?school .
?linkedinPerson linkedin:hasBSc ?school .
```

Bu örnek için başlangıç olarak seçilen ilişkili veri kümelerinin $\mathcal{Q}_{tp_1} \equiv \mathcal{Q}_{tp_2} \equiv \Delta$ olduğunu düşünelim. Şekil 5'te, $\delta_{School} \in \triangle^v$ iki bağ kümesinin hedef veri kümesi olan sanal veri kümesidir, $\lambda_m = \langle \delta_{Facebook}, \delta_{School}, \texttt{facebook:educatedIn} \rangle$ ve $\lambda_n = \langle \delta_{Linkedin}, \delta_{School}, \texttt{linkedin:hasBSc} \rangle$ şeklindedir. Kural 11'e göre, bağ kümelerinin hedef veri kümelerinin uri uzayları aynıdır, $\mathcal{L}^{space}_{\delta^{to}_{\lambda_m}} = \mathcal{L}^{space}_{\delta^{to}_{\lambda_n}} = \mathcal{L}^{space}_{\delta_{School}}$. Böylece $\delta_{Facebook}$ ve $\delta_{Linkedin}$ veri kümelerinin tp_1 ve tp_2 için dışsal ilişkili olduğu bulunmuş olur.

Son kural, Paylaşılan Değişken Çözümlemesi bakış açısı altındaki Özne Paylaşan Üçlü Desenleri Kuralı olarak adlandırılır. Adından da anlaşılacağı gibi aynı değişkeni özne olarak içeren iki üçlü desenini ele alır. Burada herhangi bir bağ kümesi tanımı çözümleme için kullanılmadığından Bağ Çözümlemesi bakış açısı kullanılmaz, bu yüzden içsel ve dışsal ilişkili olarak veri kümeleri bulunmaz. Bu çözümlemeyi incelemek için şu üçlü desenlerini örnek olarak inceleyeceğiz:

```
?city dbpprop:name 'Izmir' .
?city dc:terms ?subject .
```

Daha önceki tanımlamalarımıza göre aynı özneyi içeren üçlüler aynı veri kümelerinde bulunurlar. Buna göre Kural 12, özneleri aynı olan üçlü desenleri için ilişkili veri kümelerini, sorgulanacak veri kümelerinin kesişimini alarak bulur. Örneğe göre

üçlü desenlerinin o anki sorgulanacak veri kümelerinin $\mathcal{Q}_{tp_1} = \{\delta_{DBpedia}\}$ ve $\mathcal{Q}_{tp_2} \equiv \Delta$ olduğunu varsayalım. Özne Paylaşan Üçlü Desenleri Kuralı uygulandıktan sonra ise üçlü desenlerinin sorgulanacak veri kümeleri kümeleri $\mathcal{Q}^{new}_{tp_1} \equiv \mathcal{Q}^{new}_{tp_2} = \{\delta_{DBpedia}\}$ olur.

Kural 12 *Eğer iki üçlü deseni aynı değişkeni özne olarak içeriyorsa, bu üçlü desenlerinin sorgulanacak veri kümelerinin ortak elemanları bu üçlü desenlerinin yeni sorgulanacak veri kümeleri kümesidir.*

$\forall \delta_x, tp_i, tp_j((s_{tp_i} = s_{tp_j}) \wedge (\delta_x \in (\mathcal{Q}_{tp_i} \cap \mathcal{Q}_{tp_j})) \rightarrow \rho(\delta_x, tp_i) \wedge \rho(\delta_x, tp_j))$, $s_{tp_i} \in \mathcal{V}$ *iken.*

Bu noktaya kadar ilişkili veri kümelerini belirlemek için farklı bakış açılarıyla ilişkili veri kümesi keşif kurallarını verdik. Bu kurallar sorgunun tam çözümlemesini yapmak için beraber işletilirler. Sonraki bölüm kuralların çalışma sırasının belirlendiği çözümleme sürecini anlatmaktadır.

3.2.2 Çözümleme süreci

Yukarıda bahsedilen kurallar düzgün bir veri kümesi seçimi yapabilmek için bir arada işletilmelidir. Bu bölümde Veri Kümesi Çözümleyicisi'nin kuralları işletme süreci Algoritma 1'de gösterilmektedir. Varolan tüm veri kümeleri (Δ) , tüm bağ kümeleri (Λ) ve sorgu (bgp) veri kümesi seçim sürecinin girdileridir. Algoritma bir sorgudaki tüm üçlü desenlerinin seçilmiş kümelerini $(\mathcal{Q}_{bgp})$ döndürmektedir. Başlangıç olarak her ilişkili veri kümesi grubu varolan tüm veri kümelerini içermektedir. Daha sonra $executeSingleStepRules\,()$ ilişkisiz veri kümelerini eler. Tek adım çözümlemesi kuralları IRI-tabanlı çözümleme örneğin; sözlük eşlemesi (Kural 1 ve Kural 2), IRI'ye-bağ (Kural 2 ve Kural 5) ve IRI'nin-bağı (Kural 6 ve Kural 7) kurallarını içerir. IRI-tabanlı-çözümleme $\mathcal{Q}_{tp_i}$ 'ye bağlı olmadığından ve her işletimde aynı sonuçcu ürettiğinden bu kuralların her biri her üçlü deseni için yalnızca bir kere işletilir.

Tek adım çözümlemesindeki kuralların belirli bir sırası olmadığından her kural işletiminden elde edilen çıktıyı kullanmak üçlü desenlerinin ilişkili veri kümelerini azaltacaktır. Tek adım çözümlemesi kuralları işletildikten sonra $sendASKQueries\,()$'teki ASK sorgularını kullanarak ek bir eleme sağlanır. Her üçlü deseni için o an bulunmuş olan ilişkili veri kümelerine ASK sorguları yollanır ve ce-

Algorithm 1 Veri Kümesi Seçim Süreci

FUNCTION $analyzeDatasets\,()$

INPUT $bgp = \{tp_1, \ldots, tp_n\}$

INPUT $\Delta = \{\delta_1, \ldots, \delta_m\}$

INPUT $\Lambda = \{\lambda_1, \ldots, \lambda_t\}$

OUTPUT $Q_{bgp} = \{\mathcal{Q}_{tp_i} \mid (tp_i \in bgp) \wedge (Q_{tp_i} \subseteq \Delta)\}$

FOR EACH tp_i IN bgp

$Q_{tp_i} := \Delta$

$Q_{tp_i} := executeSingleStepRules\,(tp_i, Q_{tp_i})$

$Q_{tp_i} := sendASKQueries\,(tp_i, Q_{tp_i})$

REPEAT

$Q_{bgp}^{intermediate} := Q_{bgp}$

$Q_{bgp} := executeRepetitiveStepRules\,(bgp, Q_{bgp})$

UNTIL $Q_{bgp} = Q_{bgp}^{intermediate}$

vap alınamayan veri kümeleri elenir. Bazı üçlü desenlerinin seçilen veri kümeleri de sonraki aşamada elenir.

Diğer yandan Paylaşılan Değişken Çözümlemesi'ne dayalı kurallar birden çok üçlü desenini dikkate alır. Üçlü desenlerinin farklı kombinasyonları birbirlerinin seçilmiş veri kümesi grubunu etkilediğinden, bu kurallar üçlü desenlerinin ilişkili veri kümelerini bulmak amacıyla üçlü desenleri için o anda bulunmuş olan seçilmiş veri kümesi gruplarına bağlıdır. Bu nedenle, bu kurallar $\mathcal{Q}_{tp_i}$'nin ilişkili veri kümelerinden yeni bir veri kümesi elenmeyene kadar yinelemeli olarak işletilmektedir. $executeRepetitiveStepRules\,()$, Paylaşılan Değişken Çözümlemesi kurallarını (Kural 8, Kural 9, Kural 10, Kural 11 ve Kural 12) her üçlü deseni çiftine uygular. Q_{bgp}'deki herhangi bir seçilmiş veri kümesi grubu değişmediğinde çözümleme süreci sonlanır.

3.3 Sorgu Düzenleyici

Sorgu Düzenleyici biriminin iki sorumluluğu vardır. İlki farklı veri kümelerinden sorgulamak amacıyla harici grupları belirlemektir. Üçlü desenlerinin $(\mathcal{Q}_{bgp})$ son seçilmiş veri kümesi grupları Veri Kümesi Çözümleyicisi tarafından belirlenir ve bu işlemde kullanılır. Sorgu Düzenleyici'nin ikinci sorumluluğu ise sorgu eniyileştirme amacı ile üçlü desenleri ve üçlü deseni grupları arasındaki sıranın düzenlenmesidir. Sorgu Dü-

zenleyici'nin çıktısı SPARQL 1.1 yapısına uygun birleştirilmiş bir sorgudur [5]. Jena ARQ gibi herhangi bir sorgu motoru birleştirilmiş sorguyu işletebilmektedir.

3.3.1 Gruplama

Bu bölümde birleştirlmiş sorgunun harici gruplarının nasıl belirlendiği açıklanacaktır. Bir harici grup eg aynı veri kümesinden sorgulanacak üçlülerin bir kümesidir ve $eg \subset bgp$ şeklinde biçimselleştirilebilir. Başlangıç sorgusu sadece bir üçlü deseni kümesi iken, grup oluşturma aşamasında, Sorgu Düzenleyici sorguyu alt sorgulara ayırır ve her biri başlangıç sorgusunun ayrık bir alt kümesi olan harici grupların bir kümesini, $EG = \{eg_x, eg_y \mid (eg_x, eg_y \subset bgp) \wedge (eg_x \cap eg_y = \emptyset)\}$ oluşturur. Algoritma 2'deki grup oluşturma yöntemi, ağ maliyetini azaltmak amacıyla seçilen aynı veri kümesinden sorgulanacak üçlü desenlerini Q_{tp_i} toplayıp dışsal grupları EG oluşturur. Ancak, bu sadece gruplanacak üçlülerin ilişkili veri kümesi sayısı 1 ise, $|Q_{tp}| = 1$ mümkün olmaktadır, böylece üçlü desenlerinin sadece seçilen veri kümesinden sonuç döndürdüğü güvence altına alınmış olur. Bunun dışında eğer seçilen kümelerde birden fazla veri kümesi varsa, bu durum gruplama sonuçların kaybına sebep olabilmektedir. Çünkü gruplanmış üçlü desenleri blok olarak sorgulanmaktadır ve eğer bu veri kümeleri birbirine bağ ediyorsa (interlinking) sonuç kaybı mümkün olabilmektedir. Algoritma 2 için olası bir geliştirme bağ kümesi tanımlarını dahil ederek birden fazla veri kümesi içeren harici gruba olanak sağlamaktır.

$vars(eg)$ fonksiyonu bir üçlü deseni grubunda, eg yer alan değişkenlerin bir kümesini, $\left\{\mathcal{V}_1, .., \mathcal{V}_k\right\}$ döndürmektedir. Harici grupları oluştururken, seçilen aynı kümede iki üçlü deseni dahi olsa, bir harici grupta ortak değişken içermeyen üçlü desenleri yer almamalıdır, $vars\,(eg) \cap vars\left(\left\{tp\right\}\right) \neq \emptyset$. Çünkü bu durum gereksiz yere grubun sonuç kümesini artırmaktadır. Bu durum farklı veri kümesinden sorgulanacak üçlü desenleri, aynı veri kümesinden sorgulanacak bağlantısız üçlü desenleri arasında bir köprü oluşturduğu zaman meydana gelebilmektedir. Bu gibi durumlarda, algoritma, ara sonuçların gereksiz yere çoğalmasını engellemek için, farklı harici gruplar oluşturmaktadır.

[5] http://www.w3.org/TR/sparql11-federated-query/

Algoritma 2 Harici grupların belirlenmesi

FUNCTION $generateExclusiveGroups()$

INPUT $bgp = \{tp_1, \ldots, tp_n\}$ including n triple patterns;

INPUT $Q_{bgp} = \{Q_{tp_i} | tp_i \in bgp\}$

OUTPUT $EG = \{eg_x, eg_y \mid (eg_x, eg_y \subset bgp) \wedge (eg_x \cap eg_y = \emptyset)\}$

LET $EG := \emptyset$

WHILE $bgp \neq \emptyset$

  LET $eg := \{tp_1\}$, $j := 2$;

  WHILE $(j \leq n) \wedge (|Q_{tp_1}| = 1)$

    IF $\left(Q_{tp_1} = Q_{tp_j}\right) \wedge (vars\,(eg) \cap vars\,(\{tp_j\}) \neq \emptyset)$

      $eg := eg \cup \{tp_j\}$;

    $j := j + 1$;

  $bgp := bgp \setminus eg$;

  $EG := EG \cup \{eg\}$;

3.3.2 Sıralama

Bir SPARQL sorgusunun üçlü desenlerini sıralamak bilinen bir eniyileştirme yöntemidir (Görlitz and Staab (2011); Bernstein et al. (2007)). Böylece daha seçici üçlü desenleri (veya harici gruplar) diğerlerinden önce işlendiğinden daha küçük bir sonuç uzayı yaratır. Bu doğrultuda, WoDQA bir harici gruptaki üçlü desenlerinin sırasını ve bir sorgudaki harici grupların sırasını sezgisel tabanlı üçlü deseni seçimi karar yöntemine göre belirler.

Algoritma 3 Sıralamanın uygulanması

FUNCTION $createFederatedQuery\,()$

INPUT $EG = \{eg_x, eg_y \mid (eg_x, eg_y \subseteq bgp) \wedge (eg_x \cap eg_y = \emptyset)\}$

INPUT $Q_{bgp}^{final} = \{Q_{tp_i} \mid tp_i \in bgp\}$

OUTPUT $FQ = \left\{subQ \mid subQ = \left\langle D_{subQ}, eg^o_{subQ} \right\rangle\right\}$ where $D = \{\delta_1, \ldots, \delta_m\}$ and $eg^o = (tp_1, \ldots, tp_n)$

LET $EG^o := \emptyset$;

FOR EACH eg IN EG

  $EG^o := EG^o \cup orderTriples\,(eg)$;

$FQ := orderGroups\left(EG^o, Q_{bgp}^{final}\right)$

Algoritma 4 Sezgisel üçlü deseni sıralama

FUNCTION $orderTriples()$

INPUT $eg \subseteq bgp$

OUTPUT eg^o which is ordered list of the input eg set

LET $eg^o := sortDescending\,(eg)\,;$

LET $i := 1;$

WHILE $i < |eg|$

  LET $j := i + 1;$

  WHILE $j < |eg|$

    $j := j + 1$

    IF $vars\,(tp_i) \cap vars\,(tp_j) \neq \emptyset$

      $swap\,(tp_{i+1}, tp_j, eg^o);$

      $break;$

Sıralama sürecinin işleyişi Algoritma 3'te verilmiştir. $generateExclusiveGroups$ fonksiyonunun çıktısını girdi olarak alır ve birleştirilmiş bir sorgu FQ döndürür. FQ, her biri bir alt sorgu, $subQ$, olan, çizge deseninin üzerinde işletileceği veri kümelerinin bir kümesini [6], $D_{subQ} = \{\delta \mid (\delta_x \in Q_{tp_i}) \wedge (tp_i \in eg_{subQ})\}$, ve çizge desenini oluşturan üçlü desenlerinin sıralı bir listesini ,eg^o_{subQ}, içeren, servis çizge desenlerinin sıralı bir listesidir.

FQ'nun satır sayısı uzunluğu başlangıç temel çizge desenininkine eşit veya daha azdır, bgp, $1 \leq |FQ| \leq |bgp|$. $createFederatedQuery$, her harici gruptaki üçlüleri $orderTriples$ fonksiyonunu kullanarak sıralar ve sıralanmış kayıtları EG^o kümesine koyar. Daha sonra, $orderGroups$ fonksiyonu harici grupların sırasına karar verir ve Q^{final}_{bgp} 'yi kullanarak son birleştirilmiş sorguyu oluşturur.

Algoritma 4'teki sıralama sürecinin detaylarına $orderTriples$ fonksiyonu tanıtılarak başlanacaktır. Bu fonksiyonun girdisi bir harici gruptur , eg, ve çıktısı ise girdi kümesinin sıralı bir kaydıdır. Başlangıçta, $orderTriples$, sezgisel seçicilik değerlerini hesaplayan bir fonksiyon, $\eta : bgp \rightarrow \mathbb{Z}^+$, kullanarak verilen üçlü deseni kümesini sıralamaktadır. Bu fonksiyon üçlü deseni içerisindeki düğümlerin tiplerini ve yerlerini hesaba katarak, üçlü deseninin değerini ortaya çıkarır.

[6] Gerçekleştirimde veri kümelerinin SPARQL uç nokta adresleri SERVICE ifadesinde kullanılmıştır.

Order	Subject	Predicate	Object
1	URI	variable	URI
2	URI	URI	variable
3	URI	variable	literal
4	URI	variable	variable
5	variable	URI	URI
6	variable	URI	literal
7	variable	variable	URI
8	variable	URI	variable
9	variable	variable	literal
10	variable	variable	variable

Tablo 3.1. Üçlü çeşitlerinin sıralanması

$$\eta\left(\langle s,p,o\rangle\right)=\theta\left(s\right)\cdot 5+\theta\left(p\right)\cdot 2+\theta\left(o\right)\cdot 4$$

η fonksiyonu, verilen elemanın tipine bakarak katsayısını döndüren,θ fonksiyonundan yararlanır.

$$\theta\left(node\right)=\left\{\begin{array}{ll} node\in\mathcal{I}; & 5\\ node\in\mathcal{L}; & 3\\ node\in\mathcal{V}; & 1 \end{array}\right\}$$

$sortDescending$ üçlü desenlerini, η fonksiyonu tarafından hesaplanan sezgisel değerlerine göre sıralar. Sezgisel hesaplamaya göre belirlenmiş üçlü tiplerinin sırası Tablo 3.1'da görülmektedir. Daha sonra bu sıra üçlü desenleri arasındaki paylaşılan değişkenler dikkate alınarak değiştirilir. Paylaşılan değişken içeren üçlüleri biraraya toplamak, ilgili birleştirme işlemlerinin mümkün olduğunca erken yapılmasını sağlar ve bu da sorgu değerlendirme süresini artırır. Bu amaçla, $orderTriples$ paylaşılan değişkenleri de hesaba katarak uygun üçlü desenleri grup içerisinde yukarı kaydırır. Bu işlem sırasında belirli üçlü deseni kümesindeki (veya listesindeki) üçlü desenlerinin değişkenlerinin kümesini döndüren ve paylaşılan değişkenleri belirlemek için kullanılan $vars$ ifadesi kullanılır. Bu ifade başka algoritmalarda da kullanılacaktır.

Tüm harici gruplar , EG^o, oluşturulduktan sonra bu gruplar da $orderGroups$ fonksiyonunda ortalama sezgisel seçiciliğine göre sıralanır. Bir sıralı harici grup (eg^o)

Algoritma 5 Harici grupların sırasına karar verme

FUNCTION $orderGroups()$

INPUT $EG^o := \{eg^o \mid eg^o = (tp_x \mid tp_x \in bgp)\}$

INPUT $Q_{bgp} = \{Q_{tp_i} \mid tp_i \in bgp\}$

OUTPUT $FQ = \left\{subQ \mid subQ = \left\langle D_{subQ}, eg^o_{subQ} \right\rangle\right\}$

LET $\mathcal{F} := sortDescendingMeanHeuristic(EG^o)$;

LET $i := 1$;

WHILE $i \leq |\mathcal{F}|$

  LET $maxSharedVarCount := 0$;

  LET $eg^o_{moved} := \emptyset$;

  LET $j := i + 1$;

  WHILE $j \leq n$

    IF $\left(\left|vars\left(eg^o_i\right) \cap vars\left(eg^o_j\right)\right| > maxSharedVarCount\right)$

      $maxSharedVarCount := \left|vars\left(eg^o_i\right) \cap vars\left(eg^o_j\right)\right|$;

      $eg^o_{moved} := eg^o_j$;

  IF $(eg^o_{moved} \neq \emptyset)$

    $move\left(\mathcal{F}, eg^o_{moved}, i+1\right)$;

$FQ = constructFederatedQuery\left(\mathcal{F}, Q_{bgp}\right)$;

için ortalama sezgisel seçicilik, $\bar{eg^o}$ ile ifade edilir ve aşağıdaki formül ile hesaplanır.

$$\bar{eg^o} = \frac{\sum_{i=1}^{|eg^o|} \eta\left(tp_i\right)}{|eg^o|}$$

Ortalama seçilik değerine göre sıralama $\mathcal{F}$ ile temsil edilir. Harici grup sıralaması, gereksiz yere sonuç kümesinin artmasını engellemek için harici gruplar arasında paylaşılan değişkenleri de dikkate alır. Bu amaçla, ortalama sezgisel değer sıralamasından sonra, daha fazla ortak değişken içeren grupları arka arkaya gelecek şekilde yukarı taşır. Algoritma 5'te, $move$ ifadesi eg^o_{moved} harici grubunu $i + 1$. sıraya koyar. Son olarak, sıralı harici grup listesini ve *constructFederatedQuery* fonksiyonu tarafından seçilen kümeleri (Q_{bgp}) kullanarak servis çizge desenleri oluşturulur.

3.3.3 Diğer

Ayrıca üçlü desenlerini gruplama ve sıralamanın yanında, WoDQA'da (OptARQ) FILTER ifadesini uygun yere taşıma eniyileştirme yöntemi de gerçekleştirilmiştir. FILTER ifadesinin karar verilen yerde işletildiğinden emin olmak için, FILTER ifadesi ve bir önceki üçlü desenini içerisine alan bir grup çizge deseni oluşturulmuştur. Bundan başka WoDQA, sorgulardaki UNION ve OPTIONAL anahtar kelimelerini de destekler. Fakat, bu destekte karmaşık UNION ve OPTIONAL yapıları için iyileştirmeler gerekmektedir. Diğer yandan, GRAPH anahtar kelimesi ve boş düğümler desteklenmemektedir.

3.4 Sorgu Çalıştırıcısı

İlişkili veri kümesi seçimi ve sorgu düzenleme evreleri temel çizge deseninin birleştirilmiş halini oluşturmaktadır. Bir sorgu motorunun son sorumluluğu ise oluşturulmuş birleştirilmiş sorguyu dağıtık veri kümeleri üzerinde mümkün olduğunca hızlı çalıştırmasıdır. Aslında sorgu işletimi bir çizge üzerindeki üçlü desenlerini birbiri ile ilişkilendirmektir. Fakat birleştirilmiş sorgulamada bu durum, veri kümelerinden oluşan dağıtık çizgeler üzerindeki çizge desenlerini birbiriyle ilişkilendirmeye dönmektedir. Bu nedenle, HTTP isteklerini en aza indirgeyecek birleştirme yöntemleri çalıştırma performansını olumlu yönde etkileyecektir. Bu amaçla, WoDQA, FedX sorgu motorunun da kullandığı bağlı birleştirme yöntemini kullanmaktadır. Bağlı birleştirme, ara sonuçların topluca işlenip oluşturulan sorguların dağıtık veri kümeleri üzerinde çalıştırılması esasına dayanır. FedX'in bağlı birleştirme yöntemi SPARQL UNION kullanarak ara sonuçları alt sorgu ile topluca yollar. WoDQA'da ise bağlı birleştirme yöntemi Görlitz et. al. (Görlitz and Staab, 2011)'da bahsedildiği gibi FILTER ifadelerini kullanarak gerçekleştirilmiştir. Çünkü UNION sorgularını SPARQL cebirine dönüştürme yuvalanmış cebir ifadeleri oluşturmaktadır. Çok büyük sayıda ara sonuç olduğu zaman, çok karmaşık yuvalanmış cebir ifadesini işlemek mümkün olmamaktadır. Bu durum, ara sonuçların daha küçük gruplar halinde birkaç istek içerisinde gönderilmesini zorunlu bırakmaktadır. FILTER ifadelerini kullanma bu sorunun aşılmasını sağlayıp büyük miktardaki ara sonuçların tek istekte yollanabilmesine olanak sağlamaktadır. FILTER ifadelerini kullanmadaki bir eksik kalan yön ise, bazı SPARQL uç noktalarının FILTER ifadelerini uygun üçlü deseni ile birlikte yerleştirilmiş olmasına rağmen bunu

Algoritma 6 Bağlı Birleştirme Gerçekleştirimi

FUNCTION $executeFederatedQuery()$

INPUT $FQ = \left\{subQ \mid subQ = \left\langle D_{subQ}, eg^{o}_{subQ} \right\rangle\right\}$ where $D = \{\delta_1, \ldots, \delta_m\}$ and $eg^{o} = (tp_1, \ldots, tp_n)$

OUTPUT $B = \{b \mid b : \mathcal{V} \rightarrow \mathcal{RN}\}$

LET $B := \emptyset$

FOR EACH $subQ$ IN FQ

  LET $B^{next} := \emptyset$

  LET $bq := eg^{o}_{subQ}$

  IF $B \neq \emptyset$

    LET $b_i \in B$

    LET $cv := \left(dom\left(b_i\right) \cap vars\left(eg^{o}_{subQ}\right)\right)$

    LET $\mathcal{K} : \mathcal{L} \rightarrow B$

    FOR EACH b IN B

      $hashMapping\left(\mathcal{K}, cv, b\right)$

      $addBoundValue\left(bq, b\right)$

  FOR EACH δ IN D_{subQ}

    $B^{next} := B^{next} \cup eval\left(bq, \delta\right)$

  $B := unify\left(B, B^{next}, \mathcal{K}\right)$

dikkate almayıp tüm üçlü desenlerinden sonra en son işlemesidir. Bu eksiklik uzak uç noktadaki sorgu işletimini yavaşlatmaktadır. Bu durumun üstesinden gelmek için FILTER ifadesini uygun üçlü deseni ile birlikte, üçlü deseni FILTER'dan önce gelecek şekilde grup içerisine alınır. Böylece iyi performanslı bir bağlı birleştirme gerçekleştirimi sağlanmış olur. Bağlı birleştirme yöntemi ARQ sorgu motorunu genişleterek gerçekleştirilmiştir [7].

Algoritma 6, bağlı birleştirme tabanlı sorgu işletimi yöntemimizi tanıtmaktadır. Algoritma 6'da görülen *executeFederatedQuery* fonksiyonu girdi olarak birleştirilmiş sorguyu, FQ, alır ve çıktı olarak sonuç eşlemelerinin bir çoklu kümesini, B (binding set), döndürür. B, bağlanım kümesi, b bağlanımlarından oluşur ve her b bir *"değişken-rdf düğümü"* ikilisinden, $b : \mathcal{V} \rightarrow \mathcal{RN}$ oluşur. Algoritma boş bir B ile başlar ve daha sonra FQ'nun tüm servis çizge desenlerini, $subQ$, işleyerek sonuç eşleme-

[7] ARQ için geliştirilen Bağlı Birleştirme eklentisi http://seagent.ege.edu.tr/etmen/wodqa/bound-arq adresinden edinilebilir.

Algoritma 7 $\mathcal{K}$ çoklu-fonksiyonunun oluşturulması

FUNCTION hashMapping()

INPUT $\mathcal{K} : \mathcal{L} \rightarrow B$

INPUT $cv \subset dom\,(b)$

INPUT $b \in B$

LET $\kappa = ()$

LET $i := 1$

WHILE $i \leq |cv|$

$\kappa = \kappa \cup b\,(v_i)$

$i := i + 1$

$dom\,(\mathcal{K}) := dom\,(\mathcal{K}) \cup \kappa$, $cod\,(\mathcal{K}) := cod\,(\mathcal{K}) \cup b$

lerini oluşturur. Bir $subQ$ için ilk olarak, ara sonuçlardan bir bağlı sorgu oluşturulur ve daha sonra bu bağlı sorgu D_{subQ}'daki tüm veri kümeleri üzerinde çalıştırılır. $eval$ ifadesi, bağlı sorguyu verilen veri kümesi üzerinde çalıştırır. Gerçekleştirim olarak, bağlı değerler içeren alt sorguyu bir SPARQL uç nokta üzerinde çalıştırmayı temsil eder. B^{next}, $subQ$'nun D_{subQ}'daki tüm veri kümeleri üzerinde işletilmesiyle oluşturulmuş, sonuç eşlemelerinin çoklu kümesidir. B^{next}, sonuç B'sini oluşturmak için sonradan işlenmektedir, öyle ki B^{next}, şimdiki $subQ$'nun sonucudur ve önceki sonuç eşlemeleri ile birleştirilmesi gerekir. B ve B^{next}'in birleştirilmesi, iki kümedeki sonuçları birbirleri ile eşleyerek yeni sonuç eşlemelerinin yaratılması işlemidir. İki sonuç eşlemesini eşleştirme, aynı değişken değerlerine karşılık aynı sonuçların üretilmesi demektir, $\forall b_1 \in B, b_2 \in B^{next}, v \in (dom\,(b_1) \cap dom\,(b_2))\,(b_1\,(v) = b_2\,(v)) \rightarrow matches\,(b_1, b_2)$, ve B^{next}'teki hangi sonucun B'deki hangi sonuçtan elde edildiğini belirler. Bu tanımdaki $dom\,(b)$, SPARQL önerisinde tanımlandığı gibi b'nin alanını yani, b'deki bazı RDF terimlerine eşleşen değişkenler kümesini temsil eder.

Bağlı sorgunun oluşturulması B boş olmayana kadar, B'deki her sonuç eşlemesini dolaşarak gerçekleştirilir. Algoritma, bağlı sorgu üretilirken performansı artırmak için iki önemli iş gerçekleştirir. İlki sonuç eşlemelerini bir hash-tablosuna koyan $hashMapping$ 'tir. (Bkz Algoritma 7) Daha sonra bu hash tablosu, birleştirme işleminde karşılaştırma yapılırken kullanılır. B ve B^{next} kümelerini karşılaştırma, ara sonuçlar büyük olduğu zaman çok maliyetli olduğundan, hash tablosu kullanmak performansı önemli ölçüde artırmaktadır. Bu hash tablosu Algoritma 6'nın 8. satırındaki $\mathcal{K}$ çoklu fonksiyonu ile temsil edilir. Hash tablosunu oluşturmak için, ilk olarak hem

Algoritma 8 Bağlama değerlerinin yerleştirilmesi

FUNCTION addBoundValue()

INPUT b

INPUT $bq = (e_1, \ldots, e_n) \wedge ((e_i \in eg^o) \vee (e_i \in VNM))$

FOR EACH v in $dom\,(b)$

  LET $i := 1$

  WHILE $i \leq n$

    IF $v \in vars\,(\{tp_i\})$

      $addFilterExpr\,(bq, v, b\,(v))$

      break;

    $i := i + 1$

şimdiki servis çizge deseni, $subQ$, hem de şimdiki sonuç eşlemeleri ile B'de kullanılan ortak değişkenler belirlenmelidir. Bu değişkenler algoritmada cv (common variables) ile temsil edilmektedir ve $subQ$'nun değişkenleri $vars$ ifadesi ile elde edilmektedir. $\mathcal{K}$'nin alanı $dom(\mathcal{K})$, kayıt değerlerinin (RDF terimleri) bir kümesidir, $\mathcal{L} = \{\kappa \mid \kappa = (rn_1, \ldots, rn_n) \wedge rn \in \mathcal{RN}\}$. Bu kayıtlar sonuç eşlemelerine hızlıca erişmek için bir anahtar görevi teşkil etmektedir. Kayıtlar, sonuç eşlemeleri B ve işlenen altsorgu $subQ$ daki ortak değişkenlere, cv, göre oluşturulmaktadır. Çünkü bu değişkenlerin sonuçları B^{next}'i oluşturmak için eg^o_{subQ}'daki üçlü desenlerine geçirilecektir. Tüm sonuç eşlemeleri aynı yapıda olduğundan, ortak değişkenler,B'nin keyfi bir değişkeni olan b_i kullanılarak belirlenir. Birleştirme işlemi sırasında sonuç eşlemelerine erişimi sağladığından $\mathcal{K}$'nın değer kümesi B'dir.

İkinci performans işi de bağlı sorguyu uzaktaki veri kümesinin hızlı bir şekilde işleyeceği biçimde oluşturmaktır. Bahsedildiği üzere, bağlı sorguyu oluşturmak için FILTER ifadeleri kullanılmaktadır. Hızlı işletimi garantiye almak için, FILTER ifadeleri mümkün olduğunca önce işletilmelidir. Bu amaçla, $addBoundValue$ fonksiyonu, FILTER ifadelerini, FILTER içerisinde yer alan değişkene sahip ilk üçlü deseninin altına yerleştirir. Algoritma 8'de ifadelerin kaydı olan bir bağlı sorgu bq ile temsil edilmektedir. Bu ifadeler, bir bağlama ifadesindeki üçlü deseni de olabilmektedir (filter ifadesinin gerçekleştiriminde). VNM (variable-node list mapping), her bir elemanı, bir değişken v ve onun bağlanım değer listesini NL (node list) temsil eden bir kayıt VNL (variable-node list) olan, bağlanım ifadeleri kümesidir , $VNM = \{VNL \mid VNL = \langle v, NL\rangle \wedge NL = (rn_1, \ldots rn_n)\}$ iken $v \in \mathcal{V}$ ve

```
SELECT ?car ?brand ?modelName WHERE {
    SERVICE <:service1> { ?car :brand ?brand. }
    SERVICE <:service2> { ?brand :model ?model.
                          ?model :name ?modelName. }
}
```

Şekil 3.3. Dağıtık Örnek Sorgu

LINE	car	brand
1	:car1	:mercedes
2	:car2	:ferrari
3	:car3	:lamborghini

Tablo 3.2. `:service1`'den elde edilen ara sonuçlar

$\forall rn_i \in NL\,(rn_i \in RN).\ addFilterExpr$, eğer bulunan uygun üçlü deseninden sonra herhangi bir FILTER ifadesi yoksa, yeni bir FILTER ifadesi yaratır. Diğer durumda varolan FILTER ifadesine ekleme yapar.

Bir sorgunun bağlı birleştirme ile nasıl işletildiği Şekil 3.3'teki örnekle açıklanmaktadır. İlk alt sorgu `:service1` 'de işletildikten sonra elde edilen sonuç kümesi Tablo 3.2'de gösterilmektedir. Sonuç kümesindeki `?brand` değişkeninin bağlanımları ikinci alt sorguya yerleştirilecektir. Burada `?brand` değişkeninin bağlanım değerleri MANTIKSAL-OR ifadeleri ile FILTER bloğu içerisine koyulur. FILTER bloğu aynı değişkene sahip ilk üçlüden sonraya yerleştirilir. Daha sonra FILTER bloğu, üçlü bir blok içerisine konularak FILTER bloğunun, uygun üçlüden hemen sonra işletildiğinden emin olunmaktadır. `:service2`'ye yollanacak olan oluşturulan bağlı sorgu Şekil 3.4'de görülmektedir. Şekil 3.4'deki alt sorgu işletildikten sonra Tablo 3.3'daki sonuçlar elde edilir.

Şekil 3.4'deki değiştirilen alt-sorgu işletildiğinde Tablo 3.3'de görülen sonuçlar elde edilecektir.

İkinci alt sorgu işletildikten sonra elde edilen Tablo 3.3'deki sonuçlar Tablo 3.2'dekiler ile birleştirilir. Bu birleştirme işlemi iki sonuç kümesindeki paylaşılan değişkenler dikkate alınarak gerçekleştirilir. Örneğin, Tablo 3.3'deki ilk iki satır `brand` değişkeninin `:mercedes` bağlanımı nedeniyle elde edilmiştir ve böylelikle bu iki satır Tablo 3.2'nin ilk satırı ile birleştirilir. Birleştirme işleminden sonra elde edilen nihai

```
SELECT * WHERE{
      { ?brand :model ?model.
        FILTER ( ?brand=:mercedes ||
                 ?brand=:ferrari  ||
                 ?brand=:lamborghini )
      }
      ?model :name ?modelName.
}
```

Şekil 3.4. İkinci Alt-Sorgu İçin Oluşturulan Bağlı Sorgu

LINE	brand	model	modelName
1	:mercedes	:model1	SLS AMG Coupe
2	:mercedes	:model2	G Cross Country
3	:ferrari	:model3	458 Italia
4	:ferrari	:model4	California
5	:lamborghini	:model5	Veneno
6	:lamborghini	:model6	Sesto Elemento

Tablo 3.3. `:service2`'den elde edilen ara sonuçlar

LINE	car	brand	modelName
1	:car1	:mercedes	SLS AMG Coupe
2	:car1	:mercedes	G Cross Country
3	:car2	:ferrari	458 Italia
4	:car2	:ferrari	California
5	:car3	:lamborghini	Veneno
6	:car3	:lamborghini	Sesto Elemento

Tablo 3.4. Nihai sonuç kümesi

```
SELECT ?actor ?news WHERE {
      ?film dc:title 'Tarzan'.
      ?film lmdbvoc:actor ?actor.
      ?actor owl:sameAs ?x.
      ?y owl:sameAs ?x.
      ?y nytvoc:topicPage ?news.
}
```

Şekil 3.5. Çapraz Alan Sorgusu-4

sorgu sonuçları Tablo 3.4'da görülmektedir.

3.5 Kullanım Durumu

Bu kısımda WoDQA'nın örnek kullanımı FedBench Çapraz Alan Sorgusu 4 için veri kümesi seçimi ve düzenlenmesi anlatılarak açıklanacaktır. Şekil 3.5'teki sorgu,`'Tarzan'` başlıklı filmde oynayan aktörlerle ilgili haberleri aramaktadır.

Bu sorgu $bgp = \langle tp_1, \ldots, tp_5 \rangle$olarak temsil edilebilir.

$tp_1 = \langle$`?film`, `dc:title`, `'Tarzan'`$\rangle$,

$tp_2 = \langle$`?film`, `lmdbvoc:actor`, `?actor`$\rangle$,

$tp_3 = \langle$`?actor`, `owl:sameAs`, `?x`$\rangle$,

$tp_4 = \langle$`?y`, `owl:sameAs`, `?x`$\rangle$,

$tp_5 = \langle$`?y`, `nytvoc:topicPage`, `?news`$\rangle$.

VOID deposunda WoDQA'nın dikkate alacağı 4 adet veri kümesi olduğunu varsayalım, $\Delta = \{\delta_{geo}, \delta_{lmdb}, \delta_{nyt}, \delta_{dbp}\}$. Bu veri kümelerinin VOID tanımları ve aralarındaki bağ kümesi tanımları Bölüm 3.1'te ifade edilen temsile göre aşağıda verilmiştir.

IRI'lerin kısaltmaları Tablo 4.5'daki tanımlamalara göre yapılmıştır.

$\delta_{geo} = \langle\{\texttt{geonames}\}, \{\texttt{geovoc, dc}\}\rangle$

$\lambda_{geo2dbp} = \langle\delta_{geo}, \delta_{dbp}, \texttt{owl:sameAs}\rangle$

$\delta_{lmdb} = \langle\{\texttt{lmdb}\}, \{\texttt{lmdbvoc, dc}\}\rangle$

$\lambda_{lmdb2dbp} = \langle\delta_{lmdb}, \delta_{dbp}, \texttt{owl:sameAs}\rangle$

$\delta_{nyt} = \langle\{\texttt{nytimes}\}, \{\texttt{nytvoc, dc}\}\rangle$

$\lambda_{nyt2dbp} = \langle\delta_{nyt}, \delta_{dbp}, \texttt{owl:sameAs}\rangle$

$\delta_{dbp} = \langle\{\texttt{dbpedia}\}, \{\texttt{dbpont, dbpprop, dc}\}\rangle$

WoDQA, Algoritma 1'de anlatıldığı gibi ilişkili veri kümelerini belirlemek için, VOID tanımlamalarını çözümler. Başlangıç için her üçlü deseninin seçilen kümesi Δ'dır. Tek adım çözümleme aşamasında, tp_1, tp_2 ve tp_5'in seçilen kümeleri yüklem sözlük eşlemesi (Kural 1) ile belirlenir ve bu adımın ardından, $\mathcal{Q}_{tp_1} = \Delta$, $\mathcal{Q}_{tp_2} = \{\delta_{lmdb}\}$, $\mathcal{Q}_{tp_5} = \{\delta_{nyt}\}$şeklinde olur. tp_3 ve tp_4 için tek adım çözümlemesinde hiçbir ilişkili veri kümesi bulunamamıştır, çünkü `owl:sameAs` genel bir niteliktir ve veri kümesi tanımlamalarında `owl` bir sözlük olarak tanımlanmamıştır. $\mathcal{Q}_{tp_3}$ ve $\mathcal{Q}_{tp_4}$ hâlâ tüm veri kümelerini (Δ) içermektedir.

Tek adım çözümlemesinden sonra, üçlü desenleri için bulunan ilişkili veri kümelerinin uçnoktalarına ASK sorguları atılır. Fakat ağ maliyetinden kaçınmak için tp_3 ve tp_4 ile ilişkili veri kümesi seçimi yapılamadığından bunlar için ASK sorgusu atılmaz. Diğer üçlü desenleri için ASK sorguları işletildiğinde $\mathcal{Q}_{tp_1}$'den δ_{geo}, δ_{nyt}, δ_{dbp} veri kümeleri elenmiş ve tp_1 için δ_{lmdb} ilişkili olarak kalmıştır. tp_1'de yer alan `'Tarzan'` nesne değeri ASK sorguları atıldığında bu üçlü için önemli ölçüde eleme yapılmasını sağlamıştır. Diğer seçilen kümeler ASK sorgularından sonra değişmeden kalmıştır. $\mathcal{Q}_{tp_1} = \mathcal{Q}_{tp_2} = \{\delta_{lmdb}\}$, $\mathcal{Q}_{tp_3} = \mathcal{Q}_{tp_4} = \Delta$, $\mathcal{Q}_{tp_5} = \{\delta_{nyt}\}$.

ASK eleme aşamasından sonra yinelemeli çözümleme aşaması işletilir. Bu aşama iki üçlü deseni arasında işletilen yinelemeli kuralları kullanır ve üçlü desenleri arasındaki ilişkiler dikkate alınır. Bu aşamada tp_2 ve tp_3 üçlüleri `?actor` değişkeninden dolayı Kural 8 ve Kural 9 tarafından zincirleme üçlü desenleri olarak belirlenmiştir. Bu kurallar tp_3'ün ilişkisiz veri kümelerini elemiştir, $\mathcal{Q}_{tp_3} = \{\delta_{lmdb}\}$.

Diğer bir paylaşılan değişken de tp_3 ve tp_4 arasındaki `?x` değişkenidir. Nesne paylaşımı kuralları, Kural 10 ve Kural 11, tp_4'ün ilişkili veri kümelerini eleyerek $\mathcal{Q}_{tp_4} = \{\delta_{lmdb}, \delta_{nyt}, \delta_{geo}\}$ haline dönüştürür. $\mathcal{Q}_{tp_3}$ 'ten δ_{lmdb} ve $\mathcal{Q}_{tp_4}$ 'ten δ_{lmdb}, δ_{nyt}, δ_{geo} `owl:sameAs` yüklemi ile δ_{dbp}'ye bağlıdır.

```
SELECT ?actor ?news WHERE {
   SERVICE <http://data.linkedmdb.org/sparql> {
     ?film dc-terms:title 'Tarzan'.
      ?film lmdb:actor ?actor.
      ?actor owl:sameAs ?x. }
   SERVICE <http://data.nytimes.com/sparql> {
     ?y owl:sameAs ?x.
     ?y nytvoc:topicPage ?news. }
}
```

Şekil 3.6. Birleştirilmiş Çapraz Alan Sorgusu-4

```
SELECT * WHERE {
   { ?y owl:sameAs ?x.
     FILTER ( (?x = mpiiyago:Rosie_O%27Donnell) ||
              (?x = dbpedia:Rosie_O%27Donnell) )
   }
   ?y ny-times:topicPage ?news.
}
```

Şekil 3.7. İkinci Alt Sorgu İçin Bağlı Sorgu

y	news	x
nyt:N57399183941146195933	http://topics.nytimes.com/top/reference/timestopics/people/o/rosie_odonnell/index.html	dbpedia:Rosie_O%27Donnell

Tablo 3.5. NYTimes'tan Gelen Sonuçlar

actor	news
lmdb:actor/7	http://topics.nytimes.com/top/reference/timestopics/people/o/rosie_odonnell/index.html

Tablo 3.6. Çapraz Alan Sorgusu-4'ün Cevabı

Son olarak yinelemeli kurallar tp_4 ve tp_5 arasında bir özne paylaşımı bulmaktadır. Kural 12, $\mathcal{Q}_{tp_4}$ ve $\mathcal{Q}_{tp_5}$'in kesişimlerini alır ve tp_4 ve tp_5'in son ilişkili veri kümelerini $\mathcal{Q}_{tp_4} = \mathcal{Q}_{tp_5} = \{\delta_{nyt}\}$ olarak belirler.

Çapraz Alan Sorgusu 4'ün yeniden düzenlenmiş hali Şekil 3.6'da görülmektedir. Yeniden düzenleme sırasında üçlü desenleri için sıra değişimi gerekmemiştir. Yaratılan birleştirilmiş sorgunun biçimselleştirilmiş hali $FQ_{CD4} = \langle subQ_1, subQ_2 \rangle$, $subQ_1 = \langle \{\delta_{lmdb}\}, \{tp_1, tp_2, tp_3\} \rangle$, $subQ_2 = \langle \{\delta_{nyt}\}, \{tp_4, tp_5\} \rangle$ iken şeklindedir .

FQ_{CD4}'ün işletilmesi sırasında, $subQ_1$, LinkedMDB veri kümesinden iki çözüm eşlemeli bir sonuç kümesi üretmektedir. Bu sonuçlar Şekil 3.7'deki $subQ_2$'nin bağlı biçimine eklenir ve NYTimes veri kümesinden, Tablo 3.5'teki sonuç elde edilir. En sonunda, iki sonuç kümesinin birleşiminden sonra Tablo 3.6'daki sonuç kümesi, Çapraz Alan Sorgusu 4'ün nihai sonucu olarak oluşturulur.

4 WoDQA'NIN DEĞERLENDİRMESİ

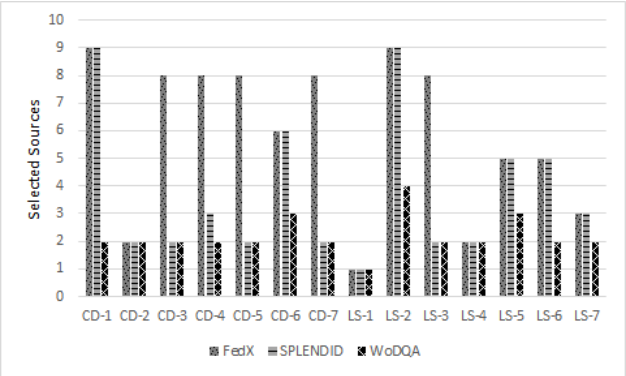
Bu bölümde WoDQA'nın performansı değerlendirilmiş ve diğer önemli yaklaşımlar FedX ve SPLENDID ile karşılaştırılmıştır. Sorgu çözümlemesi ve işletimi yöntemlerini değerlendirmek için FedBench kıyaslama kütüphanesi kullanılmıştır. İlk olarak, veri kümesi seçimini içeren ön-işleme yöntemi değerlendirilmiştir. Daha sonra bu yöntemin performansı veri kümesi sayısının arttığı durumda incelenmiştir. Son olarak bağlı birleştirmenin performansı ile doğrudan alakalı olan çalıştırma performansı için sorgu işletim süreleri ve işletim sırasında atılan HTTP isteği sayısı diğer yaklaşımlar ile karşılaştırılmıştır.

4.1 Kıyaslama Kütüphanesi Kurulumu

Diğer yaklaşımların değerlendirme sonuçları da FedBench kıyaslama kütüphanesine göre elde edildiğinden WoDQA'yı diğer yaklaşımlar ile karşılaştırabilmek için bu kütüphane kullanıldı. FedBench, birleştirilmiş sorgu motoru performansını değerlendirmek için oldukça uygun ve geniş bir sorgu alanı sunar. Tüm Cross Domain (Çapraz Alan) ve Life Sceinces (Hayat Bilimleri) sorguları değerlendirmede kullanılmıştır. Bu sorgular için gerekli veri kümelerinin yığınları (Cross Domain için DBpedia, NYTimes, LinkedMDB, SW-Dogfood, Jamendo ve GeoNames veri kümeleri; Life Sceinces için KEGG, Drugbank, ChEBI ve DBpedia veri kümeleri) indirilerek 4store RDF sunucusuna yüklendi ve her bir veri kümesi birer SPARQL uçnokta olarak açıldı. FedBench'in kullandığı bu veri kümeleri gerçek hayatta FedBench'ten beklenenler ile aynı sonuçları vermemektedir, bu nedenle bu veri kümelerinin yığınları FedBench'ten indirilerek yerel bir sunucuya yüklenmek durumunda kalındı. Diğer yandan en uygun bağlı birleştirme performansını belirlemek için FedBench sorguları hem Virtuoso hem de 4store üzerinde denendi. Değerlendirmenin gerçekleştirildiği bilgisayar donanımı özellikleri AMD FX(tm)-6100 Six-Core Processor × 6 (3.3 Ghz), 8 GB RAM, Ubuntu 12.04 64-bit OS'tur.

4.2 Ön-işleme Yönteminin Değerlendirilmesi

Bir birleştirilmiş sorgu motorunun performansını değerlendirirken ilişkili veri kümelerini seçmek için, çözümlemenin yapıldığı ön-işleme yönteminin başarısını değerlen-



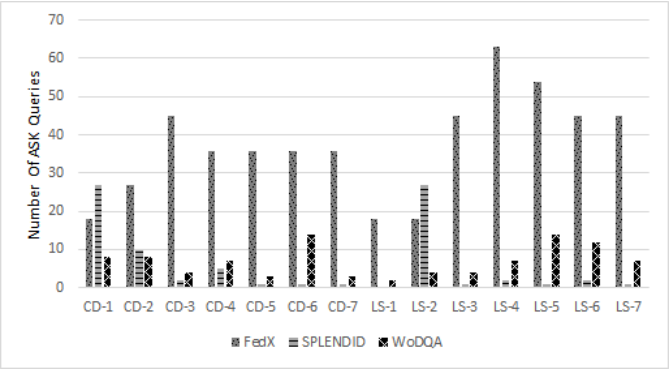
Şekil 4.1. Seçilmiş Veri Kümesi Sayıları

dirmek önemlidir. WoDQA veri kümelerini seçerken en az sayıda veri kümesini sorgulayarak tam sonuç elde etmeyi hedeflemektedir. Bu nedenle WoDQA'nın ön-işleme yöntemi gerçekte sonuç döndürmeyecek veri kümelerini bulup elemeye dayanmaktadır. Ön işleme yöntemini diğer sorgu motorları ile karşılaştırmak için seçilen veri kümesi sayısı ve ASK sayısı kıyaslamaları bu bölümde ele alınmıştır.

Şekil 4.1 WoDQA'nın veri kümesi seçiminin FedX ve SPLENDID ile kıyaslanmasını temsil etmektedir. Bu karşılaştırmaya göre FedX'in seçim yöntemi en kötü olarak göze çarpmaktadır. FedX herhangi bir üst veri kullanmamaktadır ve her üçlü deseni için ilişkili veri kümelerini, varolan tüm veri kümelerine ASK sorgusu atarak belirlemektedir. Bu seçim yöntemi, sadece yüklemi bağlı olan ve `owl:sameAs` gibi her veri kümesinde çokça rastlanabilecek bir yüklem içeren üçlü desenleri için zayıf kalmaktadır.

Şekil 4.1'de görüldüğü gibi, SPLENDID'in veri kümesi seçim yöntemi FedX'ten daha iyidir. SPLENDID'in PT-GSA[1] yöntemi, kendi değerlendirmeleri içerisinde en iyi sonuçlara sahip olduğundan performansı kıyaslamak için bu yöntemin sonuçları dikkate alınmıştır. SPLENDID'in veri kümesi seçim mekanizması sınıf ve yüklem indekslemeye (VOID tanımlamalarına dayanarak) ve ayrıca ASK sorgularına dayanır. Yüklem ve sınıf indeksleme SPLENDID'in seçim yöntemini CD3, CD4, CD5, CD7, LS3 sorguları için FedX'ten daha iyi yapmaktadır. Çünkü bu sorgular, içerdikleri "`?x owl:sameAs ?y`" gibi üçlülerde `owl:sameAs` yüklemine sahiptir, bu yüzden FedX bu üçlüleri tüm uçnoktalara sormaktadır ve neredeyse tüm veri kümeleri `owl:sameAs` yüklemine sahiptir. Daha sonra FedX `owl:sameAs` yük-

[1]PT-GSA, "predicate and type index and grouping of sameAs patterns" ifadesinin kısaltmasıdır. Görlitz vd. tarafından ilgili yayında Şekil 4'te açıklanmıştır.



Şekil 4.2. ASK Sayıları

lemine sahip bir üçlü içeren tüm veri kümelerini ilişkili olarak seçer. Fakat SPLENDID `owl:sameAs` yüklemine sahip ve aynı bağlı olmayan değişken değerine sahip üçlüleri gruplayarak bunların aynı veri kümelerinden sorgulanmasını sağlar. Böylece CD3, CD4, CD5 sorgularında `owl:sameAs` yüklemi içeren üçlü desenleri için, aynı bağlı olmayan değişken içeren üçlüleri gruplamadan dolayı, ilişkili veri kümelerinde bir eleme sağlanmış olur.

Şekil 4.1, sözlük ve URI uzayı üstverisine dayalı olan ve ayrıca üçlü desenleri arasındaki ilişkileri de dikkate alan WoDQA'nın başarılı seçim yöntemini ortaya koymaktadır. Özellikle CD4, CD6, LS2, LS5, LS6 ve LS7 sorguları için seçilen veri kümesi sayısı diğer sorgu motorlarına kıyasla çok daha iyidir. CD4, CD6 ve LS6 sorgularında özne paylaşımı kuralı iyi seçim sağlamaktadır. Çünkü aynı özne değişkenine sahip üçlü desenlerinin ilişkili veri kümelerinin kesişimi alınarak, iki üçlü deseni için bu kesişimden arta kalan veri kümeleri ilişkili olarak belirlenmektedir. LS2 ve LS5 sorguları için zincirleme eşlenen üçlüler kuralı ilişkisiz olanları elemektedir. Zincirleme olarak eşleşen üçlü desenleri arasında bağ kümesi olup olmadığına bakılmaktadır. Son olarak LS7 sorgusunda ise nesne paylaşımı kuralı daha iyi bir seçim sağlamaktadır.

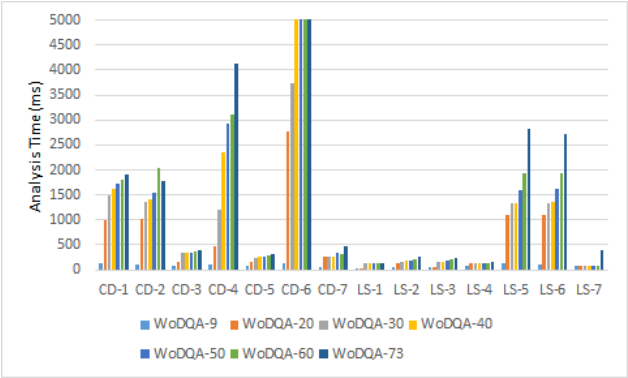
Seçim yöntemini değerlendirmenin bir diğer bakış açısı ise ASK sorgularıdır, öyle ki karşılaştırmaya dahil edilen yaklaşımlar ASK sorgularını kendi seçim süreçlerine dahil etmişlerdir. Büyük birleştirme işlemlerinde ASK sayısı uzun çözümleme sürelerine neden olmaktadır. ASK sorgularını önbelleklemek mümkün olmasına rağmen, bu önbelleği değişken ve büyük veri bulutlarında yönetmek zor olmaktadır. Bu değerlendirmeyi yapabilmek için FedX ve SPLENDID'in çözümleme algoritmaları kendi çalışmalarında (Schwarte et al., 2011b; Görlitz and Staab, 2011) bahsedildiği gibi bire bir gerçekleştirilmiştir.

	FedX	SPLENDID	WoDQA
CD-1	18	27	8
CD-2	27	10	8
CD-3	45	2	4
CD-4	36	5	7
CD-5	36	1	3
CD-6	36	1	14
CD-7	36	1	3
LS-1	18	0	2
LS-2	18	27	4
LS-3	45	1	4
LS-4	63	2	7
LS-5	54	1	14
LS-6	45	2	12
LS-7	45	1	7

Tablo 4.1. ASK Sayıları

Şekil 4.2 ve Tablo 4.1, en büyük ASK sayısının FedX'in olduğunu ortaya çıkarmaktadır, çünkü seçim yöntemi tüm üçlü desenlerini tüm uç noktalara sorma esasına dayanmaktadır. WoDQA, FedX'e göre çok daha az ASK sayısına sahiptir, çünkü sözlük ve URI uzayı üst verisini kullanarak sağladığı eleme ile ASK sorgusu atılacak uç noktaların sayısını azaltmaktadır. Bununla birlikte WoDQA'nın ASK sayısı SPLENDID'ten yalnızca CD1, CD2 ve LS2 sorgularında azdır. Bu sorgularda SPLENDID'in ASK sayısı konusundaki zayıflığı bağlı olmayan yüklem (değişken) içeren üçlü desenleridir. Diğer yandan SPLENDID'in yüklem dizini, diğer sorgular için çok daha az ASK sayısı üretmektedir. SPLENDID, bağlı yükleme sahip üçlü desenleri için ilişkili veri kümelerini doğrudan dizinden almaktadır, ancak sözlükler ilişkili veri kümelerini doğrudan belirleyemediği için, daha iyi eleme yapmak amacıyla WoDQA bu üçlüler için ASK sorguları yollar. Bu sebeple WoDQA, FedX'ten az, SPLENDID'ten çok ASK sorgusu yollamaktadır. Bu durum, tasarım açısından dinamik ortamlarda, yüklem dizinini güncelleme ile sözlük elemesinden dolayı yollanan ASK sorguları arasında bir ödün verme olarak görülebilir.

Son olarak sorgunun tekrar düzenlenmesinden bahsetmek faydalı olur. Bu sü-



Şekil 4.3. ASK Önbelleklemesiz Çözümleme Süreleri

reç ön işlemeyi tamamlar ve sorgu değerlendirme performansını artırmayı amaçlar. Tüm sorgular için harici gruplar belirlenir. Sezgisel tabanlı düzenleme CD2, CD5, CD6, CD7 ve LS4 sorgularının işletim planlarını değiştirerek sorgu işletimini etkiler. CD5 sorgusu için de harici grupların sırası değişmektedir. Bu gruplama ve sıralama işlemleri olmadan FedBench sorguları için iyi işletim sürelerine ulaşmak mümkün olmamaktadır.

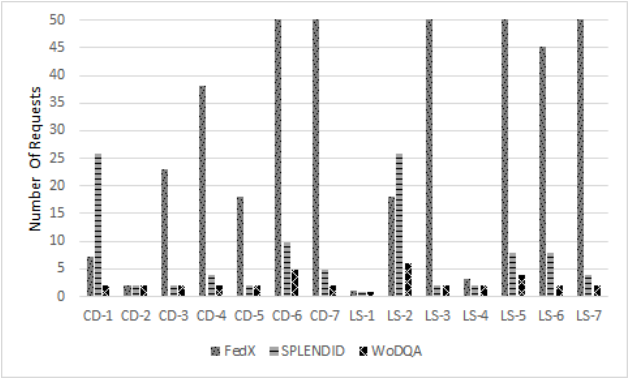
4.3 Veri Kümesi Sayısının Artması Durumu

Ön-işleme yönteminin performansı en çok ASK sorgularından etkilenir. ASK sorgularından kaynaklanan performans kaybından korunmak için WoDQA, FedX'in yaptığı gibi ASK sorgularının sonuçlarını önbellekler. ASK sorgularının çözümleme zamanındaki etkisini göstermek için, 9'u FedBench veri kümesi olmak üzere, LOD bulutundaki 73 veri kümesinin[2] VOID'i oluşturuldu. (VOID belgelerini oluşturduktan sonra kapanan veri kümelerinin yerel kopyaları değerlendirme için hazır hale getirilmiştir.) Daha sonra veri kümeleri, ASK sonuçlarını önbelleklemeden değerlendirmek için restgele seçildi. Tablo 4.2 farklı veri kümesi sayısı için zaman değerlerini göstermekte ve Şekil 4.3 çözümleme zamanı sonuçlarının değerlendirmesinin kıyaslamasını göstermektedir. Şekil 4.3'teki CD1, CD2, CD4, CD6, LS5 ve LS6 sorguları için çözümleme sürelerinde, birleşime yeni veri kümeleri eklendikçe gözle görülür bir artış olmaktadır. CD1 ve CD2'nin yavaşlığının sebebi, bu sorguların nesne konumunda DBpedia URI'sini içermeleridir ve birçok veri kümesi DBpedia sözlüğünü kullanmaktadır. CD4 "`?subject owl:sameAs ?object`" şeklinde, CD5 "`?subject foaf:`

[2]http://dsi.lod-cloud.net tarafından yayınlanan uç noktalara ulaşarak elde edilen VOID belgeleri http://seagent.ege.edu.tr/wodqa/evaluation.html adresinden edinilebilir.

	9	20	30	40	50	60	73
CD-1	133	998	1485	1622	1725	1800	1906
CD-2	99	1017	1356	1418	1545	2049	1772
CD-3	72	148	331	341	350	376	393
CD-4	104	482	1215	2356	2916	3102	4129
CD-5	75	149	247	255	256	299	323
CD-6	131	2764	3750	6738	8810	9476	11731
CD-7	46	252	250	258	329	327	469
LS-1	20	34	126	126	124	132	127
LS-2	53	140	151	184	184	204	254
LS-3	46	64	161	165	173	204	226
LS-4	79	128	128	134	133	134	152
LS-5	120	1099	1335	1347	1591	1935	2828
LS-6	115	1106	1341	1358	1621	1938	2712
LS-7	74	87	92	92	84	82	403

Tablo 4.2. ASK Önbelleklemesiz Çözümleme Süreleri



Şekil 4.4. İstek Sayıları

`name ?object`" şeklinde, LS5 ve LS6 "`?subject dc:title ?object`" şeklinde üçlüler içermektedir. Öyle ki bu sözlükler yaygındır ve veri kümelerinin çoğu tarafından kullanılmaktadır. Bu durum da veri kümesi sayısı arttıkça sorgu çözümlemesi sürelerinin yavaşlamasına neden olmaktadır. Bu yüzden, ASK sayısı artar ve ayrıca bazı veri kümelerinin uç noktaları çok yavaştır.

4.4 Bağlı Birleştirmenin Değerlendirilmesi

WoDQA bağlı birleştirme yöntemini, sorgu birleştirme yöntemine dahil etmiştir. Birleştirme yöntemi, birleştirmeyi gerçekleştirirken atılan istek sayısını belirlemektedir. HTTP isteği sayısını azaltan bir birleştirme yöntemi kullanmak sorgu motoru performansını artırmaktadır.

Şekil 4.5'te FedX, SPLENDID ve WoDQA sorgu değerlendirme zamanı cinsinden karşılaştırılmıştır. Tablo 4.4'te görünen sonuçlar çözümleme ve işletim sürelerini birlikte içermektedir. FedX'in tablodaki değerleri Schwarte et. al. (Schwarte et al., 2011b)'den alınmıştır ve SPLENDID'in değerleri de of Görlitz et. al. (Görlitz and Staab, 2011)'in resimlerden tahminlenerek alınmıştır. Sorgu değerlendirme performansını yorumlamadan önce, sorguları çalıştırma sırasında atılan HTTP isteği sayısı için de bir kıyaslama tanımlanmıştır. İstek sayısı değerlendirmesi Tablo 4.3'te ve karşılaştırma Şekil 4.4'te görülmektedir. Diğer sorgu motorlarının istek sayıları kendi çalışmalarından alınmıştır.

Veri kümesi seçimi yöntemi ve UNION tabanlı bağlı birleştirme algoritması sebebiyle FedX'in istek sayısı performansı en kötüsüdür. (Bkz. Şekil 4.4) RDF sunucuları genellikle bir SPARQL sorgusundaki UNION sayısını kısıtlarlar. Bu sebeple,

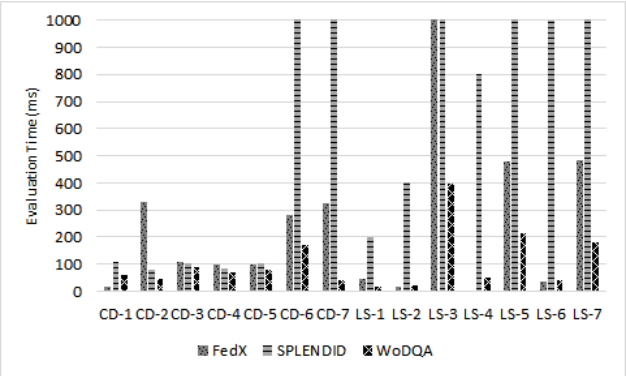
	FedX	SPLENDID	WoDQA
CD-1	7	26	2
CD-2	2	2	2
CD-3	23	2	2
CD-4	38	4	2
CD-5	18	2	2
CD-6	185	10	5
CD-7	138	5	2
LS-1	1	1	1
LS-2	18	26	6
LS-3	2059	2	2
LS-4	3	2	2
LS-5	458	8	4
LS-6	45	8	2
LS-7	485	4	2

Tablo 4.3. İstek Sayıları

ara sonuçların ayrılması ve birden fazla HTTP isteği ile yollanması gerekmektedir. Bu durum FedX'in CD6, CD7, LS3, LS5, LS7 sorguları için istek sayısını 50'nin üzerine taşımaktadır. Diğer yandan SPLENDID, birleştirme için WoDQA'dan daha çok HTTP isteğinin atılmasına gereksinim duymaktadır. Büyük miktarda ara sonuç değeri üreten sorgular için bağlı birleştirme, küçük miktarda ara sonuç değeri üreten sorgularda ise hash birleştirme yöntemi kullanmaktadır. Fakat CD1, CD6, LS2, LS5 ve LS6 sorguları için istek sayıları başarısız veri kümesi seçiminden dolayı yüksektir.

WoDQA seçim yöntemi ve bağlı birleştirme yöntemi sayesinde FedBench sorgularını en az istek sayısı (bkz. Şekil 4.4) ile ve en az sürede (bkz. Şekil 4.5) çalıştırmaktadır. Veri kümesi seçimi performansı için değerlendirme Bölüm 4.2'de verilmişti. Bu bölümdeki değerlendirmeler ise bağlı birleştirme gerçekleştiriminin başarısını ortaya koymaktadır. WoDQA CD2, CD6, CD7, LS3, LS5, ve LS7 sorguları için en iyi değerlendirme sonuçlarını vermektedir ve tüm sorguları Şekil 4.5'de görüldüğü gibi 400 milisaniyeinin altında çalıştırmaktadır. FedX LS3 sorgusu için, SPLENDID ise CD6, CD7, LS3, LS5 ve LS7 sorguları için 1 saniye limitini aşmıştır .

Bağlı birleştirme gerçekleştirilirken tüm ara sonuçların tek bir istekte yollanması amaçlandı. Bu amaçla, SPARQL sorgu dilindeki çeşitli özelliklerin bağlı birleştirme için kullanılması denendi. Bu özellikler UNION, FILTER IN, FILTER OR ve VALUES özellikleridir. Bu gerçekleştirimler hem 4store hem de Virtuoso RDF saklayıcılarında denendi. İlk seçenek UNION gerçekleştirimiydi, fakat UNION blokları derin bir yuvalanmış cebir ifadesi yarattığından, bu gerçekleştirim iki RDF sunucusunda da başarısız olmuştur. FedX bu yöntemi ara sonuçları parçalayıp, bu parçalanmış sonuç kümelerinin her birini ayrı istekte yollamaktadır. Bununla birlikte, bu yöntem LS3 gibi çok büyük sayıda ara sonuç içeren sorgular için sorun teşkil etmektedir. İkinci gerçekleştirim ise CD6, LS3, LS5, ve LS7 sorgularında başarısız olmuş olan FILTER IN yaklaşımıdır. İki sunucunun da sorgu işlemcileri FILTER IN ibaresi için eniyileştirilmemiştir. Üçüncü deneme de VALUES ibaresinin kullanımıdır. Şimdilik sadece Virtuoso sunucusu, VALUES ibaresini desteklemektedir, fakat Virtuoso'nun VALUES bloğu için sonuç kümesi sayısında kısıtlamaları vardır. Bu sebeple VALUES yöntemi tüm FedBench sorguları için kullanılamamaktadır. Son olarak, FILTER OR gerçekleştirimi için Virtuoso, özellikle LS3 sorgusunda bariz olarak görülen kısıtlamalara sahiptir. Bağlı birleştirmede FILTER OR yöntemi kullanılıp sorguları 4store üzerinde çalıştırarak Tablo 4.4'daki hızlı çalıştırma sürelerine ulaşılmıştır. 4store'un



Şekil 4.5. Sorgu İşletim Zamanları

sorgu işlemcisi ve derleyicisi en iyi performansı sağladı. Aslında VALUES ibaresinin anlamsallığı bağlı birleştirme için çok daha uygundur. Çünkü VALUES'ta önceden tanımlanmış değerler üçlü deseni işletilmeden önce bağlanır, FILTER'da ise bu değerler üçlü deseni işletildikten sonra uygulanır. RDF depoları VALUES ibaresini geniş ve büyük bloklar için dikkatlice ele almalıdır, çünkü bağlı birleştirme sorgu birleştirmenin performansını artırmaktadır.

Bölüm 3.4'de bahsedildiği gibi uzak uçnoktalardaki ata bağlanımları çocuk bağlanımlar ile eşlemek için hash tabloları kullanılmaktadır. Bu gerçekleştirim karşılaştırma yükünü ortadan kaldırmakta ve böylece ara sonuçların $O(n)$ karmaşıklığında işlenmesini sağlamaktadır. Bu da WoDQA'nın bağlı birleştirmesinin değerlendirme sonuçlarındaki başarısının bir başka sebebidir.

	FedX	SPLENDID	WoDQA
CD-1	15	110	63
CD-2	330	80	49
CD-3	109	103	88
CD-4	100	85	72
CD-5	97	101	79
CD-6	281	10000	172
CD-7	324	3000	42
LS-1	47	200	20
LS-2	16	400	24
LS-3	1470	20000	399
LS-4	1	800	51
LS-5	480	21000	217
LS-6	34	1000	44
LS-7	481	20000	183

Tablo 4.4. Sorgu İşletim Zamanları

void:	<http://rdfs.org/ns/void#>
rdf:	<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
owl:	<http://www.w3.org/2002/07/owl#>
dc:	<http://purl.org/dc/terms/>
foaf:	<http://xmlns.com/foaf/0.1/>
dbpedia:	<http://dbpedia.org/resource/>
dbpont:	<http://dbpedia.org/ontology/>
dbpprop:	<http://dbpedia.org/property/>
lmdb:	<http://data.linkedmdb.org/resource/>
lmdbvoc:	<http://data.linkedmdb.org/resource/movie/>
geonames:	<http://www.geonames.org/>
geovoc:	<http://www.geonames.org/ontology#>
nytimes:	<http://data.nytimes.com/>
nytvoc:	<http://data.nytimes.com/elements/>
mpiiyago:	<http://mpii.de/yago/resource/>
drugont:	<http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
drugbank:	<http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/>
drugcategory:	<http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory/>
kegg:	<http://bio2rdf.org/ns/kegg#>
bio2rdf:	<http://bio2rdf.org/ns/bio2rdf#>

Tablo 4.5. Önek Tanımları

5 ÖNBELLEKLEME

5.1 Özet

Bağlı veri bulutu üzerinde çok sayıda veri kümesi ve bu veri kümelerinin sorgulanabildiği uçnokta bulunmaktadır. Bu uçnoktaların tipleri bağlı oldukları veri kümesine göre değişiklik gösterebilmekte ve her birinin kısıtlamaları ve erişilebilir olma süresi değişebilmektedir. Bağlı veri bulutunun bu değişken ve düzensiz yapısı sebebiyle veriyi kaynağından sorgulamak her zaman mümkün olamayabilmektedir. Bu durum hizmet kalitesini (Quality of Service) azaltmakta ve sorgulama sonucunda istenilen veri elde edilemeyebilmektedir. Bu durumu aşmanın bir yolu veriyi kopyalayıp yerelde sorgulamaktır (Rietveld, 2012; Schandl, 2010), ancak burada kopyalama maliyeti ve güncellik sorunu karşımıza çıkmaktadır. Önbellekleme, hizmet kalitesini sağlayan diğer bir yöntem olarak görülebilir. Sorgulanan veri kümesi erişilemez durumda olsa bile, sorgu bu veri kümesinde daha önce işletilmiş ise, sonucu önbellekte yer alacağı için elde edilebilmektedir. Böylece sonuç kaybı yaşanmadan sorgulama gerçekleştirilebilmekte ve sorgunun uçnokta üzerinde çalıştırma maliyeti de ortadan kalktığı için sorgulama da çok hızlı bir şekilde gerçekleştirilmiş olmaktadır.

Veri kümeleri sorgulanırken, ilgili uçnoktalara ortak sonuçlara sahip sorguların atılma olasılığı yüksektir. İlk akla gelecek yöntem her sorgunun sonucunu doğrudan önbelleğe almak olacaktır. Ancak önbelleğin doluluğu önbellekteki uygulama nesneleri ile doğrudan ilgili olduğundan (Martin et al., 2010) ve her yeni sorgu sonucu önbelleğe alındığında bu sonuçların önbellekte bulunup bulunmama durumuna bakılmadığından, sorgular ortak sonuçlar içerse dahi bu ortak sonuçlar önbellekte farklı bir nesneymiş gibi tutulmaktadır. Bu durumda önbellek aynı nesneler tarafından şişirilerek etkin şekilde yönetilememiş olmaktadır. Önbelleklemenin kullanıldığı uygulama boyunca genellikle aynı ve benzer sorgular atıldığından, önbellek mantıksal olarak aynı olan fakat farklı bir nesneymiş gibi davranılan nesnelerin bir çöplüğü durumuna gelmiş olur. Bu durum da önbellekte daha az sorgu sonucunun tutulmasına böylece de önbellek ıskalarının artmasına, bu da sorgulama süresinin uzamasına neden olmaktadır. Ortak sorgu sonuçları tekrar önbelleğe alınmayacak şekilde tutulduğunda, önbellek hem boyut hem de sorgu kapasitesi bakımından etkin şekilde yönetilmiş olur.

Önbelleklemede karşılaşılan bir diğer çok basit ama hayati problem ise fiziksel

olarak (değişken isimleri) farklı ancak mantıksal olarak aynı olan sorgulardır. Bilindik önbellek mimarisinde sadece değişken isimleri farklı olan, mantıksal olarak aynı sorguların cevapları önbelleğe alınacağı zaman bu sorgulara farklı sorguymuş gibi davranılır. Bu durumda yeni gelen sorgunun cevapları aslında önbellekte yer alıyorken, farklı sorguymuş gibi görülecek, sonuçlar önbellekten getirilmeyip uçnoktadan sorgulanacak ve ayrıca yeni gelen bu mantıksal eş sorgunun varolan cevapları da önbellekte yeni bir girdi olarak tutulacaktır. Ortak olan sorgu sonuçları ele alındığında bu yeni sorgunun sonuçları önbellekte varolan bir sorgu ile aynı olacağından yeni nesneler yaratılıp önbellek doldurulmamış olur. Ancak önbelleğin kapasitesi eğer önbellekte tutulacak girdi sayısına göre belirlenmiş ise bu durum fazladan aynı sorgu-cevap girdisini önbelleğe almak demektir. Bu durumda hem işletim zamanından ödün verilerek boş yere bir sorgu işletilecek, hem de önbellek iyi yönetilememiş ve fazladan aynı sorgu-cevap ikilisi önbellekte tutulacaktır. Mantıksal olarak aynı olan sorguları ele alıp işleyebilen bir mekanizma kurulduğunda hem gereksiz yere sorgulama maliyetinden, hem de önbellekte fazla sorgu tutarak etkin yönetilememesi durumundan kaçınılmış olacaktır.

5.2 Ön Tanımlar

5.2.1 Altyapı

Sparql Önbellekleme: Bir SPARQL sorgusunun önbelleklemesi, sorguyu uçnokta(lar) üzerinde çalıştırma maliyetinden kurtulup sonuçları doğrudan önbellekten getirmeyi amaçlar. Önbellekte genellikle sık tekrar eden veya en son işlenen sorgular yer almaktadır, böylece sorgu sonuçları önbellekte yer alıyorsa hızlıca elde edilebilmektedir.

Önbellek İsabeti: Bir SPARQL sorgusunun sonucunun, sorgulandığı anda önbellekte yer alması durumudur. Böylece sorgu uçnokta üzerinde işletilmeye gerek olmadan sonucu elde edilebilmektedir.

Önbellek Iskası: Atılan sorgunun sonucu önbellekte yok ise önbellek ıskası olur. Bunun sonucunda sorgu uçnoktadan sorgulanır ve cevabı önbelleğe alınır.

TTL (Time to Live - Yaşam Süresi): Önbellekteki bir sorgu sonucunun önbellekte yer alabileceği en yüksek süredir. Bir sorgu sonucu en çok erişilen veya en son

zamanda erişilen sorgu dahi olsa yaşam süresi doldu ise önbellekten atılır.

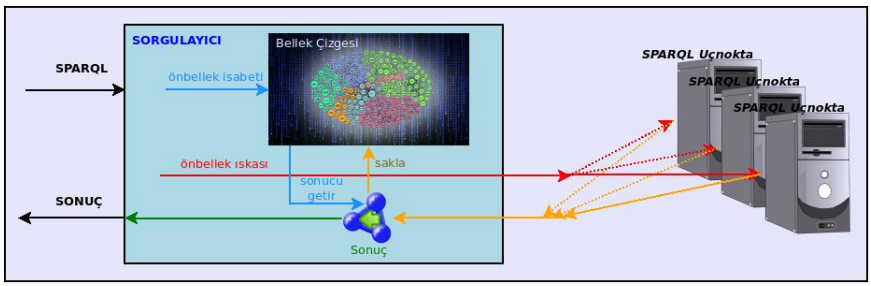
TTI (Time to Idle - Atıllık Süresi): Bir sorgunun önbellekte erişilmeden kalabileceği en yüksek süredir. Atıllık süresini aşan sorgu bayatlamış olur ve önbellekten bir sorgu tahliye edileceği zaman bayatlamış olan bu sorgu öncelikli olarak tahliye edilir.

Pareto (Güç Yasası) Dağılımı: Bu teoriye göre yoğun olarak tekrar eden sorgular tüm sorguların sadece küçük bir kısmıdır. Sorguların geri kalan büyük kısmı ise az tekrar etmektedir. Bu durumda aynı sorgu gelecekte yüksek olasılıkla tekrar edecektir. 80/20 kuralı olarak da bilinir, buna göre uçnoktaya gelen sorgu isteklerinin %80'ini, özgün sorguların %20'si oluşturmaktadır.

5.2.2 Sistem modeli

Bu bölümde gerçekleştirilen SPARQL önbellekleme mimarisinin işleyişine genel hatlarıyla değinilecektir. Sistem modeli Şekil 5.1'de görülmektedir. İlk olarak gelen SPARQL sorgusu sorgulayıcı modülü tarafından karşılanır. Sorgunun cevabının bellek çizgesinde yer alıp almadığı kontrol edilir. Eğer cevap bellek çizgesinde yer alıyor ise önbellek isabeti (cache hit) olmuş demektir ve bu durumda sonuç bellek çizgesinden getirilerek kullanıcıya döndürülür. Diğer senaryoda ise sorgu sonucu bellek çizgesinde yer almamaktadır. Bu durumda önbellek ıskası durumu gerçekleşir ve sorgunun çalıştırılması gerekir. SPARQL sorgusu, tekil sorgu biçiminde veya birleştirilmiş sorgu biçiminde olabilmektedir. Buna göre sorgu, yapısı gereği tekli veya çoklu uçnokta üzerinde sorgulanarak cevabı elde edilir. Elde edilen cevap bellek çizgesine kaydedilir ve en sonunda da kullanıcıya döndürülür.

Sistemin genel yapısı gereği iç içe çalışan iki ana modülden oluştuğu görülmektedir. Bunlar sorgulayıcı modül (querier) ve bellek çizgesi modülüdür. Sorgulayıcı modülü sistemin genel işleyişini yönetir ve bellek çizgesine erişme ve onu yönetme yetkisine sahiptir. Gerektiğinde bellek çizgesinden cevapları alır, gerektiğinde ise sorguyu uçnokta(lar) üzerinden sorgulayarak cevapları bellek çizgesine kaydeder. Ayrıca sorguların cevaplarının bellek çizgesinde, üçlüler şeklinde saklanacak şekilde oluşturulmasını sağlar. SPARQL sorgulamaya önbelleği dahil ederek, geliştirilmiş bir sorgulama yeteneği sunar. Bellek çizgesi modülü, önbelleğin yer aldığı ve bakımının yapıldığı kısımdır. Üçlüler halinde elde edilen SPARQL sorgu cevaplarının, belleğin et-



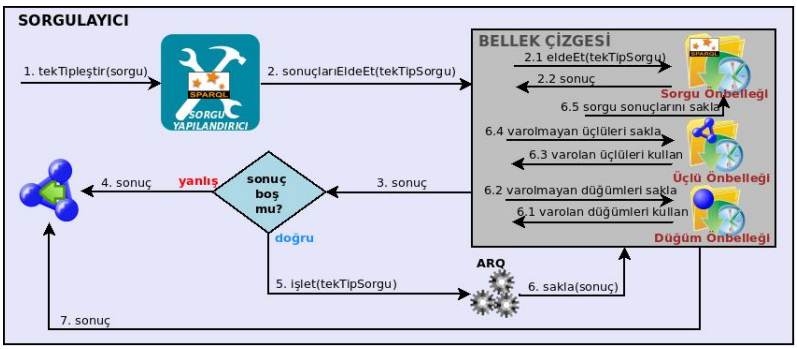
Şekil 5.1. SPARQL Önbellek Sistem Modeli

kin şekilde kullanılarak kaydedilmesi sağlanır. Şekil 5.2'de ayrıntılı olarak görüldüğü üzere bellek çizgesinde; sorgu önbelleği, üçlü önbelleği ve düğüm önbelleği olmak üzere 3 aşamalı önbellek yapısı bulunmaktadır. Sorgu önbelleğinde sorgular ve sorguların cevabı olan üçlüler yer alır. Üçlü önbelleği ve düğüm önbelleği bellek yönetimini sağlar. Aynı üçlü ve düğümlerin yeni bireylerinin yaratılması engellenip varolan bireyleri kullanılır, böylece önbelleğin gereksiz yere israf edilmesinden kaçınılmış olur.

5.3 SPARQL Önbellekleme Gerçekleştirimi

Tezin bu kısmında SPARQL önbellekleme mimarisi, bu mimarideki yapılar ve bu yapıların işleyişi detaylandırılarak anlatılacaktır. Önbellekleme yaklaşımında, SPARQL sorgularının sonuçlarını üçlüler şeklinde elde etmek ve her bir üçlü bireyini yalnızca bir kez kullanarak bellekte fiziksel bir çizge oluşturup önbelleği etkin olarak yönetmek amaçlanmıştır. Sorgu cevaplarının üçlüler şeklinde tutulabilmesi için ya sorguların cevaplarının üçlüye dönüştürülmesi ya da sorgunun construct sorgusuna dönüştürülmesi gerekmektedir. Bu mimaride SELECT sorgularının cevaplarını üçlü olarak elde etmek için sorgular eşdeğer CONSTRUCT sorgusuna dönüştürülerek cevapların üçlü olarak elde edilmesi sağlanmaktadır. ASK sorgularında ise sonuç üçlü şeklinde ifade edilerek bellek çizgesine uygun yapıya dönüştürülmektedir. Tüm sorgu tiplerinin cevapları üçlüler halinde tutularak, önbellek kararlı bir çizge yapısında oluşturulmuş olur. Oluşan bu çizge bellek yapısı, bağlı verinin üçlü doğasına uygundur ve SPARQL önbellekleme için önbelleğin etkin yönetildiği bir çözüm sunmaktadır.

Önbellek mimarisinde sorgular, Bellek Çizgesi birimi ile etkileşim halinde olan Sorgulayıcı birimi ile sorgulanır. Sorgulayıcı birimi, sorguların önbellekten veya uçnoktadan elde edilmesine karar veren ve sorguların cevaplarının çizge bellek mimarisine uygun şekilde kaydeden birimdir. Bölüm 5.3.1'de Sorgulayıcı birimindeki yapılar



Şekil 5.2. Önbelleklemeli Sorgu Çalıştırma Süreci

ve birimin çalışma mekanizması anlatılmaktadır.

SELECT sorguları, Sorgu Düzenleyici Birimi kullanılarak CONSTRUCT sorgusuna dönüştürülerek, sonuçları çizge belleğe kaydedilmeye uygun hale getirilir. Bu birim aynı zamanda sorguların tek tip yapılmasını sağlayarak, yalnızca değişken isimleri farklı mantıksal olarak aynı olan sorguların farklı sorguymuş gibi önbellekte tutulmasını önlemiş olur. Bölüm 5.3.2'de birimin işlevselliklerinden bahsedilmektedir.

Sorgular ve cevapları Bellek Çizgesi birimi'nde tutulur. Önbelleğin yer aldığı kısımdır. Cevaplar üçlüler halinde ve her bir üçlünün yalnızca tek bir bireyi (instance) kullanılacak şekilde saklanır. Sorgu önbelleği, üçlü önbelleği ve düğüm önbelleği olmak üzere üç aşamalı önbellek yapısı bulunur ve bunlar bellek çizgesi yapısının oluşturulmasını sağlar. Bellek çizgesi, Bölüm 5.3.3'te detaylı olarak anlatılmaktadır.

Tahliye Birimi önbelleğin bakımını sağlar. Önbellekte yer alan bir sorgunun yaşam süresinin bitmesi, en az kullanılan durmuna düşmesi vs. gibi durumlarda gerekli sorgunun önbellekten tahliye edilmesini sağlar. Böylece sorgu ve bu sorgunun cevabını oluşturan üçlüler, önbellekten başka sorgular için sonuç kaybı oluşturmayacak şekilde tahliye edilir. Önbelleğin bakımını yapan bu birimin işleyişi Bölüm 5.3.4'te verilmiştir.

5.3.1 Sorgulayıcı birimi

Bu bölümde geliştirilen önbellekleme mimarisinde, SPARQL sorgularını çalıştırmakla görevli Sorgulayıcı biriminden bahsedilecektir. Sorgulayıcı birimi diğer tüm birimleri sarmalar ve mimarinin algoritması burada çalıştırılır, sonuçların kaydedilmesi aşamasında Bellek Çizgesi birimine erişip, onu kullanma yetkisine sahiptir. Algoritma 9'daki *executeQuery()* fonksiyonunda bir SPARQL sorgusunun Sorulayıcı birimi ile çalıştırılması anlatılmaktadır. Bu fonksiyon girdi olarak bir SPARQL sorgusu, SQ

alır ve çıktı olarak bu sorgunun cevabını, R_{SQ} döndürür. Çalıştırılacak olan SPARQL sorgusu, SQ ilk olarak Sorgulayıcı'ya gelir. SQ, Sorgu Yapılandırıcı kullanılarak, $standardize(SQ)$ metodu ile tek tip hale getirilerek, tek tipleştirilmiş sorgu, SQ^s elde edilir. Bu işlem, yalnızca değişken isimleri bakımından farklı, mantıksal olarak aynı sorguları yakalayabilmek ve bu sorguların önbellekte bir kez yer almasını sağlamak için uygulanmaktadır. Sorgu tek tip hale geldikten sonra, cevabı olan önbelleklenmiş üçlü listesi, CT, bellek çizgesi, MG'den $getResultTriples(MG, SQ)$ metodu ile elde edilmeye çalışılır. Eğer üçlü listesi sorgu önbelleğinde yer alıyorsa, sorgunun işletilmesine gerek yoktur, böylece $turnTriplesIntoResult(CT)$ metodu ile doğrudan Bellek Çizgesi'nden elde edilen üçlü listesi, CT, sorgu sonucu, R_{SQ}'ye dönüştürülür. Cevap bellek çizgesinde yer almıyorsa sorgunun işletilmesi gerekmektedir. Sorguyu uçnokta(lar) üzerinde işletmek için Jena ARQ sorgu motoru kullanılmaktadır. $execute(SQ)$ metodu ile sorgu işletilerek sonucu, R_{SQ} elde edilir. Eğer sorgu SELECT sorgusu ise sonuç result set olacağından, sonuçları üçlüler şeklinde elde etmek adına, Sorgu Yapılandırıcı ile CONSTRUCT haline dönüştürülür. Sorgu eğer ASK sorgusu ise normal şekilde işletilerek boolean sonuç değeri elde edilir. Boolean değeri "true" ise sorgulanan sorgunun cevabının olduğunu, "false" ise olmadığını belirten bir üçlü oluşturulur. CONSTRUCT ve DESCRIBE sorgularının sonucunda ise RDF modeli nesnesi döner ve Jena aracı ile modelde yer alan üçlüler kolayca elde edilebilir. $turnResultIntoTriples(R_{SQ})$ metodu ile sorgu sonucu, R_{SQ} üçlü listesi, CT'ye dönüştürülür ve bellek çizgesine kaydedilir.

Sonuçlar kullanıcıya döndürülürken, üçlülerin sorgu sonucuna dönüştürülmesi gerekmektedir ve bu işlem $turnTriplesIntoResult(CT)$ metodu ile gerçekleştirilir. Buna göre üçlüler önce model içerisine alınır. Sorgu CONSTRUCT veya DESCRIBE ise model nesnesi doğrudan döndürülür. Sorgu SELECT ise model üzerinde çalıştırılır ve kullanıcıya result set nesnesi döndürülür. Sorgunun ASK sorgusu olduğu durumda ise cevap tek bir üçlü nesnesidir ve içerilip içerilmeme durumuna göre boolean değeri döndürülür.

5.3.2 Sorgu yapılandırıcı birimi

Bu bölümde Sorgu Yapılandırıcı Birimi'nin temel işlevsellikleri anlatılacaktır. Burada temel olarak çeşitli amaçlarla SPARQL sorgusu ile ilgili bölme, dönüştürme ve kontrol gibi işlemler gerçekleştirilmektedir. Bu birimin gerçekleştirdiği en önemli iki temel

Algoritma 9 Sorgunun Sorgulayıcı birimi ile işlenmesi

FUNCTION executeQuery()

INPUT $SQ \in FQ \vee SQ \in UQ$ WHERE $FQ = \{subQ \mid subQ = \langle D_{subQ}, eg^{o}_{subQ} \rangle\}$ AND $UQ = \langle \delta, eg \rangle$

OUTPUT R_{SQ}

LET $MG = \left\langle QC, TC, NC \right\rangle$

LET $QC = \left\{ \mathfrak{el} | \mathfrak{el} = \left\langle SQ, CT \right\rangle \right\}$ WHERE $CT = \left\{ ctp_1, .., ctp_n \right\}$

LET $ctp = \left\langle tp, Q \right\rangle$ WHERE $Q = \left\{ SQ_1, .., SQ_m \right\}$

LET $TC : tp \rightarrow ctp$

LET $NC : nd \rightarrow cnd$

$SQ^s = standardize(SQ);$

$CT = getResultTriples(MG, SQ^s);$

$if(CT = null)$

$R_{SQ} = execute(SQ);$

$CT = turnResultIntoTriples(R_{SQ});$

$storeTriples(MG, CT);$

$R_{SQ} = turnTriplesIntoResult(CT);$

işlevsellik, *SELECT-CONSTRUCT* dönüşümü ve tek tipleştirme işlemidir.

İlk olarak *SELECT-CONSTRUCT* dönüşümü işleminden bahsedilecektir. *SELECT* sorgusunun sonucu result set olarak elde edilir ancak gerçekleştirilen çizge bellek yapısında sonuçların üçlü olarak kaydedilmesi gerekmektedir. Bu nedenle *SELECT* sorgusu *CONSTRUCT* sorgusuna dönüştürülerek cevabın result set değil de RDF modeli olarak elde edilmesi sağlanır. Daha sonra elde edilen bu RDF modeli oluşturan üçlüler kolayca Bellek Çizgesi'ne kaydedilebilmektedir. *CONSTRUCT* edilmek üzere, dönüşüm işleminde sorguyu oluşturan tüm üçlüler elde edilir. Sorgunun *PREFIX*'leri ve *WHERE* kısmı tekrar kullanılmak üzere ayrıştırılır. *WHERE* kısmında yer alan üçlüler *CONSTRUCT* içerisine yazılarak *"**SELECT ***"* yapısına eşlenik bir *CONSTRUCT* yapısı oluşturulur. Son olarak *PREFIX*, *CONSTRUCT* ve *WHERE* kısımları birleştirilerek *SELECT* sorgusu, eşleniği olan bir *CONSTRUCT* sorgusuna dönüştürülmüş olur.

Birimde gerçekleştirilen bir diğer önemli işlevsellik sorgu tek tipleştirmedir. Sorgulayıcı birimine gelen SPARQL sorguları önbelleğe kaydedileceği zaman veya önbellekte yer alıp almadıkları sorgulanacağı zaman tek tipleştirilmeleri gerekir. Çünkü iki sorgunun yalnızca değişkenleri farklı, ancak sorgular mantıksal olarak aynı olabilir. Bu durumun basit bir örneği Tablo 5.1'de görülmektedir. Tablodaki iki sorgu mantıksal olarak aynı, sözdizimsel olarak ise özne ve nesne değişkenleri açısından farklılık göstermektedir. Anlamsallık bakımından SORGU-1'deki **"?person**" değişkeni, SORGU-2'deki "***?subject***" değişkeni ile, SORGU-1'deki "**?friend**" değişkeni ise SORGU-2'deki "***?object***" değişkeni ile anlamsal açıdan aynıdır. Bu iki sorgu önbelleğe doğrudan kaydedilecek olursa iki farklı sorguymuş gibi önbellekte yer tutacaktır. Ancak sorgulara uygulanacak tek tipleştirme işlemi ile her sorgu yalnızca tek bir tipe dönüşeceğinden, önbellekte gereksiz yer tutmalar olmayacaktır.

Sorgu tek tipleştirme Algoritma 10'da görülmektedir. Tek tipleştirme işlemi için önce sorgudaki tüm değişkenler V elde edilir. Elde edilen değişkenler sırayla tek tipleştirilmeye başlar. Değişken sırası i'ye göre önce mutlak dizin (absolute index) ai , daha sonra ise sıra (order) o belirlenir. Harf değerleri İngilizce'deki 26 harf $L = \{l_1, ..l_{26}\}$ içerisinden seçilir. Örnek olarak SORGU-1'i Algoritma 10 ile tek tipleştirilecektir. Sorgunun değişkenleri elde edildiğinde $V = \{v_1 := person, v_2 := friend\}$ değerini alır. İlk olarak $person$ değişkeninin mutlak dizini $ai = 1 \bmod 26 = 1$ olarak elde edilir. Buna göre $l_1 = a$ harfi seçilecektir. Daha sonra değişkenin sırası $o = 1 \div 26 = 0$ olarak belirlenir. a harfi 0 sayısı ile birleştirilerek $person$ değişkeninin yeni değeri $a0$

Algoritma 10 Sorgu Tek Tipleştirme

FUNCTION standardize()

INPUT $SQ \in FQ \vee SQ \in UQ$ WHERE $FQ = \left\{ subQ \mid subQ = \left\langle D_{subQ}, eg^{o}_{subQ} \right\rangle \right\}$ AND $UQ = \langle \delta, eg \rangle$

OUTPUT SQ^s

LET $L = \left\{ l_1, ..l_{26} \right\}$

LET $i := 1$;

LET $ai := 0$;

LET $o = 0$;

$SQ^s = SQ$;

$V = vars(SQ)$;

WHILE $i < \left| V \right|$

$ai = i \bmod \left| L \right| - 1$;

$o = i \div \left(\left| L \right| - 1 \right)$;

$vc = l_{ai} \cup o$;

$SQ^s = replace(SQ^s, v_i, vc)$;

SORGU-1	SORGU-2	SORGU-3
SELECT * WHERE{?person foaf:knows ?friend}	SELECT * WHERE{?subject foaf:knows ?object}	SELECT * WHERE{?a0 foaf:knows ?b0}

Tablo 5.1. Yalnızca Değişkenleri Farklı Sorgular

olur. Son adımda sorgudaki $person$ değişkeni $a0$ değişkeniyle değiştirilir. Benzer olarak $friend$ değişkeninin mutlak dizini $ai = 2 \bmod 26 = 2$, harf değeri $l_2 = b$ ve sırası $o = 2 \div 26 = 0$ olarak belirlenir, böylece $b0$ değişkeni oluşturulur. $friend$ değişkeni $b0$ değişkeni ile yer değiştirir ve Tablo 5.1'deki SORGU-3 elde edilmiş olur. Aynı şekilde SORGU-2 de Algoritma 10'a göre tek tipleştirildiğinde SORGU-3 elde edilecektir.

5.3.3 Bellek çizgesi birimi

Bu bölümde önbelleğin bulunduğu ve yapısının yönetildiği Bellek Çizgesi biriminden bahsedilecektir. Bellek Çizgesi birimi 3 aşamalı bir önbellek yapısına sahiptir. Bu birimler Şekil 5.2'de görülen Sorgu Önbelleği(Query Cache), Üçlü Önbelleği(Triple Cache) ve Düğüm Önbelleği(Node Cache)dir.

- **Sorgu Önbelleği:** Sorgu-Cevap ikilileri şeklinde sorgu cevaplarının tutulduğu yerdir. Sorgu Önbelleği'nin yönetimi Ehcache[1] önbellekleme aracı kullanılarak sağlanmıştır. Ehcache sayesinde, sorgu önbelleği ilklendirilirken önbelleğin yönetim işlevleri olan Yaşam Süresi(TTL), Atıllık Süresi(TTI), tahliye yöntemi; en az kullanılan(LFU), en eski kullanılan(LRU), ilk giren ilk çıkar (FIFO), önbelleğe alınacak en yüksek eleman sayısı, önbelleğin boyutu vb. gibi yapılar,

[1] http://ehcache.org/

basit bir yapılandırma dosyası ile veya kod içerisinden kolayca ayarlanabilmektedir. Böylece önbelleğin bakımı basit ve güvenli bir şekilde yapılmış olmaktadır. Sorgu, önbelleğe alınmaya uygun olacak şekilde sarmalanır. Daha sonra sorgunun cevabı olan üçlü listesinin önbelleğe kaydedilmeye uygun hale getirilmesi işlemine geçilir. Her bir üçlünün önbelleğe kaydedilmeden önce üçlü önbelleğinde yer alıp almadığına bakılır. Eğer üçlünün önceden saklanmış bir bireyi (instance) üçlü önbelleğinde varsa, varolan bu nesne kullanılır. Aynı şekilde üçlüleri oluşturan düğümler de önbelleğe kaydedilmeden önce düğüm önbelleğinde olup olmadığı denetlenir. Eşdeğer düğüm varsa, düğüm önbelleğinde yer alan nesne kullanılır. Üçlü önbelleğinde yer almayan üçlüler üçlü önbelleğine ve düğüm önbelleğinde yer almayan düğümler de düğüm önbelleğine kaydedilir. Böylece bellekte tekil bir fiziksel çizge oluşturularak, etkin bir önbellek yönetimi sağlanmış olur. Bu şekilde sorgu ve cevabı olan üçlü listesi, önbelleğe alınmaya uygun hale getirildikten sonra, sorgu önbelleğine kaydedilir.

- **Üçlü Önbelleği:** Üçlü önbelleğinde TC, sorgu önbelleğindeki tüm sorguların cevapları olan üçlüler yer almaktadır. Her üçlü; özne, yüklem ve nesne düğümlerinden oluşmaktadır ve üçlü önbelleğine kaydedilmeden önce düğümlerinin düğüm önbelleğinde, NC olup olmadığı kontrol edilir. Öncelikle önbelleklenmemiş düğümler düğüm önbelleğine kaydedilir. Sonra düğüm önbelleğinde varolan düğümler ve yeni kaydedilen düğümler kullanılarak, yeni bir üçlü oluşturulur. Böylece üçlülerin kullanımında düğüm seviyesine kadar denetlenerek, aynı düğümlerin bireylerini boş yere saklama yükünden kaçınılmış olunur. Üçlü, tp ve ilişkili olduğu sorgular, Q_{tp} sarmalanıp önbelleklenmiş üçlü nesnesi, ctp yaratılarak üçlü önbelleğine kaydedilmektedir. Temel olarak bir üçlü ile şifrelenmiş önbelleklenmiş üçlü şeklindeki bir hashmap $TC : tp \rightarrow ctp$ yapısındadır.Üçlü havuzu olarak düşünülebilir; her üçlü, havuzda yalnızca bir kez yer alacak şekilde toplanır, üçlünün bağlı olduğu tüm sorgular sorgu önbelleğinden tahliye edildikten sonra, üçlü havuzdan çıkarılır.

- **Düğüm Önbelleği:** Önbelleğin en atomik yapısı olan düğümler yer alır. Düğüm önbelleğinde, NC üçlüleri oluşturan tüm özne, yüklem ve nesne düğümleri toplanır. Elemanları üçlü önbelleğine benzer olarak, düğüm ile şifrelenmiş önbelleklenmiş düğüm şeklindeki bir hashmap $NC : nd \rightarrow cnd$ yapısında tutulmak-

tadır. Üçlüleri oluşturan düğümler burada yer alıyorsa, varolan düğüm üçlünün yapısında kullanılmak üzere düğüm önbelleğinden getirilir. Varolmayanlar ise düğüm önbelleğine kaydedildikten sonra kullanılır. Her düğüm nd, kendisiyle ilişkili olan sorgular, Q_{nd} ile sarmalanarak önbelleklenmiş düğüm cnd, nesnesi yaratılır ve yalnızca bir tane düğüm bireyi önbellekte yer alacak şekilde düğüm önbelleğine kaydedilir. Düğüm önbelleğindeki bir düğümün ilişkili olduğu tüm sorgular sorgu önbelleğinden tahliye edildikten sonra, o düğüm de düğüm önbelleğinden çıkarılır.

Sorgu Önbelleği, Üçlü Önbelleği ve Düğüm Önbelleği'nin çalışma şekillerinden bahsettikten sonra, Algoritma 11'daki *store()* fonksiyonunda görüldüğü üzere, genel olarak bir sorgunun ve cevabı olan üçlülerin tasarlanan önbellek mimarisinde nasıl saklandığından bahsedelim. Fonksiyon girdi olarak sorgu nesnesi, SQ ve cümle(statement) listesi, S almaktadır. Bellek çizgesine bir sorgu, SQ ve cevabı olan cümleler, S kaydedilmek üzere geldikten sonra, cevabı oluşturan cümle listesi, varolan üçlüleri yeniden kullanmak ve varolmayan üçlüleri de Üçlü Önbelleği'ne kaydetmek üzere dolaşılır. Öncelikle cümle, st üçlüye, tp dönüştürülür. Daha sonra, Algoritma 12'ta anlatılan *storeNode()* fonksiyonu ile bu üçlünün özne, yüklem ve nesne düğümlerinin Düğüm Önbelleği, NC'de olup olmadığı kontrol edilerek, yer almayanlar $createCachedNode(nd, SQ)$ fonksiyonunda SQ ile sarmalanarak kaydedilir, yer alanların ilişkili sorguları Q_{cnd} güncellenip SQ eklenir ve düğüm saklanacak üçlünün yapısında yeniden kullanılır. Böylece üçlünün bir kısmı veya tamamının varolan düğümlerden oluşturulması sağlanabilir. Yeniden kullanılabilir bu üçlü nesnesi elde edildikten sonra Algoritma 13'da açıklanan *storeTriple()* fonksiyonunda, üçlünün Üçlü Önbelleği'nde yer alıp almadığına bakılır. Eğer Üçlü Önbelleği'nde zaten varsa, üçlünün ilişkili sorguları, Q_{ctp} güncellenerek yeni sorgu SQ eklenir ve varolan üçlü kullanılmak üzere getirilir, yok ise $createCachedTriple(tp, SQ)$ fonksiyonunda üçlü, ilişkili sorgu SQ ile sarmalanarak, Üçlü Önbelleği'ne kaydedilir ve cevap üçlüleri, CT'ye eklenir. Bu adım sorgunun cevabı olan tüm üçlüler için tekrarlandıktan sonra sorgunun cevabı olan üçlüler yenilenmiş ve Sorgu Önbelleği'ne alınmaya hazır hale getirilmiştir. Son adımda sorgu, cevabı olan üçlüler ile kapsüllenerek, el Sorgu Önbelleği, QC'ye kaydedilir.

Bellek Çizgesi, sorgu cevabı kaydetmenin yanında, varolan bir sorgunun cevabını getirme, bir sorgu cevabının, bir üçlünün veya bir düğümün önbellekte yer alıp

Algoritma 11 Cevabın Bellek Çizgesi'ne kaydedilmesi

FUNCTION storeTriples()

INPUT $SQ \in FQ \vee SQ \in UQ$ WHERE $FQ = \{subQ \mid subQ = \langle D_{subQ}, eg^{o}_{subQ} \rangle\}$ AND $UQ = \langle \delta, eg \rangle$

INPUT $S = \left\{ st_1, .., st_n \right\}$

LET $QC = \left\{ \mathfrak{el} | \mathfrak{el} = \left\langle SQ, CT \right\rangle \right\}$ WHERE $CT = \left\{ ctp_1, .., ctp_n \right\}$

LET $TC : tp \rightarrow ctp$

LET $NC : nd \rightarrow cnd$

FOR EACH st IN S

$tp = turnStatementIntoTriple(st);$

$subject_{tp} = storeNode(subject_{tp}, SQ);$

$predicate_{tp} = storeNode(predicate_{tp}, SQ);$

$object_{tp} = storeNode(object_{tp}, SQ);$

$tp = createTriple(subject_{tp}, predicate_{tp}, object_{tp});$

$ctp = storeTriple(tp, SQ);$

$CT = CT \cup ctp;$

$\mathfrak{el} = \left\{ SQ, CT \right\};$

$QC = QC \cup \mathfrak{el};$

almadığını kontrol etme gibi işlevlere sahiptir.

5.3.4 Tahliye birimi

Önbellek mimarisindeki sorgu sonucu, üçlü ve düğüm tahliye işlemlerini yöneten Tahliye Birimi bu bölüm kapsamında açıklanacaktır. Tahliye birimi 3 temel sınıftan oluşmaktadır, bunlar EvictionManager, Evicter ve EvictionListener sınıflarıdır. Önbellekten bir sorgu tahliyesi olması durumunda tahliye olan sorgunun cevabı olan üçlülerin ve bu üçlüleri oluşturan düğümlerin de tahliye edilmesi gerekmektedir. Ancak her üçlü ve düğüm bellekte yalnızca tek bir kez yer aldığından, tahliye edilecek üçlü veya düğüm başka bir sorgu ile de ilişkili ise, önbellekte tutulur ve yalnızca tahliye edilen sorgu ile ilişkisi kesilir. Şimdi sıra ile bu yapıların tahliye işlemini nasıl yönettiği Algoritma 14'da anlatılan *evict()* fonksiyonu ile desteklenerek açıklanacaktır.

Sorgu Önbelleği'nde Ehcache aracının Cache nesnesi kullanılmaktadır ve bu önbellek üzerinde gerçekleştirilen tüm işlemlerin sonucunda bir event fırlatılmakta-

Algoritma 12 Düğümün düğüm önbelleğine kaydedilmesi

FUNCTION storeNode()

INPUT $nd \mid nd \in tp$

INPUT $SQ \in FQ \vee SQ \in UQ$ WHERE $FQ = \{subQ \mid subQ = \langle D_{subQ}, eg^{o}_{subQ} \rangle\}$ AND $UQ = \langle \delta, eg \rangle$

OUTPUT $nd \mid nd \in tp$

LET $NC : nd \rightarrow cnd$

$cnd = getCachedNode(NC, nd)$;

IF $cnd = null$

$cnd = createCachedNode(nd, SQ)$;

ELSE

$Q_{cnd} = Q_{cnd} \cup SQ$;

$NC = NC \cup cnd$;

$nd = getNode(cnd)$;

Algoritma 13 Üçlünün üçlü önbelleğine kaydedilmesi

FUNCTION storeTriple()

INPUT tp

INPUT $SQ \in FQ \vee SQ \in UQ$ WHERE $FQ = \{subQ \mid subQ = \langle D_{subQ}, eg^{o}_{subQ} \rangle\}$ AND $UQ = \langle \delta, eg \rangle$

OUTPUT ctp

LET $TC : tp \rightarrow ctp$

$ctp = getCachedTriple(TC, tp)$;

IF $ctp = null$

$ctp = createCachedTriple(tp, SQ)$;

ELSE

$Q_{ctp} = Q_{ctp} \cup SQ$;

$TC = TC \cup ctp$;

Algoritma 14 Bellek Çizgesi'nden tahliye algoritması

FUNCTION evict()

INPUT $\mathfrak{el}_{evicted} = \left\langle SQ, CT \right\rangle$

LET $TC : tp \rightarrow ctp$

LET $NC : nd \rightarrow cnd$

LET $NM : int \rightarrow nd$

$SQ = getQuery(\mathfrak{el}_{evicted})$;

$CT = getTriples(\mathfrak{el}_{evicted})$;

$NM = evictTriples(TC, CT, SQ)$;

$evictNodes(NC, NM, SQ)$;

dır. EvictionListener sınıfı bu event'lerden tahliye ile ilgili olanı yakalar ve tahliye edilen eleman elde edilmiş olur. Daha sonra tahliye edilen elemanı işlemesi için yeni bir Evicter nesnesi yaratılır. Evicter nesnesi java.lang.Runnable arayüzünü gerçekleştirmektedir ve Thread mantığı ile çalışmaktadır. Evicter sınıfı, Üçlü Önbelleği ve Düğüm Önbelleği'ne erişme hakkına sahip olan ve run() metodunda tahliye edilen sorgu ile ilgili gerekli üçlülerin ve düğümlerin tahliyesini gerçekleştiren ve böylece de Bellek Çizgesi'nin bakımını tamamlayan önemli bir parçadır. Algoritma 14'e göre Evicter sınıfında, tahliye edilen eleman, $\mathfrak{el}_{evicted}$ elde edildikten sonra, bu elemanı oluşturan sorgu, SQ ve önbelleklenmiş üçlü listesi, CT parçaları ayrıştırılır. $evictTriples(TC, CT, SQ)$ fonksiyonunda üçlüler tek tek kontrol edilerek, SQ'den başka sorguyla ilişkili olup olmadığına bakılır. Eğer başka sorguyla ilişkili değilse $tp_{evicted}$, üçlü önbelleği, TC'den tahliye edilir, eğer üçlü başka sorgu ile de ilişkili ise yalnızca tahliye edilen sorgu $SQ_{evicted}$ üçlünün ilişkili sorgularından çıkarılır. Üçlü tahliye işlemi sırasında, tahliye edilen bu üçlüleri oluşturan düğümler belirlenerek düğüm haritası, NM'ye kaydedilir. $evictNodes(NC, NM, SQ)$ fonksiyonunda da üçlüleri oluşturan düğümler tahliye edilir. Tahliye edilecek düğüm, tahliye edilen sorgudan başka sorgularla da ilişkiliyse, düğüm tahliye edilmez, sadece tahliye edilen sorguyla olan ilişiği kesilir. Eğer tahliye edilen düğüm, tahliye edilen sorgudan başka sorguyla ilişkili değilse $nd_{evicted}$, düğüm önbelleği NC'den tahliye edilir. Evicter nesnesi EvictionManager nesnesi ile çalıştırılarak tahliye işlemi uygulanır. EvictionManager, Evicter nesnesinin pararlel olarak çalışmasını sağlamaktadır ve böylece önbellek işlemlerinde aksama ve süre kaybı olmadan, tahliye işlemi arka planda gerçekleştirilmiş olur.

6 ÖNBELLEKLEME DEĞERLENDİRMESİ

Tezin bu bölümünde Bellek Çizgesi mimarisinin performansı 3 aşamada değerlendirilmiştir. İlk aşamada Berlin SPARQL Benchmark kıyaslama kütüphanesi genişletilerek önbelleğin genel yetenekleri ortaya konmuştur. İkinci aşamada FedBench sorguları önce ARQ ile, daha sonra Bellek Çizgesi ile çalıştırılmıştır ve çalıştırma süreleri karşılaştırılmıştır. Son aşamada Bellek Çizgesi, WoDQA'ya bütünleştirilerek WoDQA'nın FedBench sorgularını çalıştırma süreleri ortaya konmuştur. Değerlendirmenin gerçekleştirildiği bilgisayar donanımı özellikleri AMD FX(tm)-6100 Six-Core Processor × 6 (3.3 Ghz), 8 GB RAM, Ubuntu 13.04 64-bit OS'tur.

6.1 Tekil Uçnoktalı Sorgularda Değerlendirme

Bu kısımda ilk olarak uçnoktaların performansını değerlendirmeyi amaçlayan Berlin SPARQL Kıyaslama Kütüphanesi'nin kurulumundan ve önbellekleme performansını değerlendirmek üzere nasıl uyarlandığından bahsedilecektir. Daha sonra önbelleğin tek uçnokta üzerinde çalıştırılan BSBM e-ticaret sorguları üzerindeki performansı çeşitli kriterlere göre değerlendirilecektir.

6.1.1 Berlin sparql kıyaslama kütüphanesinin genişletilmesi

Berlin SPARQL Kıyaslama Kütüphanesi, uçnoktaların sorgu çalıştırma performansını değerlendiren bir araçtır. BSBM ile değerlendirilecek olan veri kümesine, otomatik olarak üretilen seçilen miktarda üçlü yüklenir. Daha sonra üçlülerin yüklendiği bu veri kümesinin performansı, BSBM'nin içerisinde yer alan kullanım durumlarından biri ile, arka arkaya sorgular atılarak ölçülür. BSBM'de sorgu taslakları kullanılır ve her yeni sorgu yollanacağı zaman veri kümesinde üretilmiş halde bulunan değerlerden bir veya birkaçı taslaktaki yerine konulur. Uçnoktanın değerlendirilmesinde kullanılan bu sorguların tamamına yakını birbirinden farklıdır. Ancak gerçekte uçnoktaya bu kadar farklı sorgunun yollanması gerçekçi değildir.

Genişletilen BSBM kütüphanesi Pareto Dağılımı'na göre çalışacak şekilde düzenlenmiştir. Pareto Dağılımı'na göre sorguların tekrar eden büyük çoğunluğu, tüm sorguların yalnızca küçük bir kısmıdır. Sorguların kalan büyük kısmı ise az tekrar etmektedir. Buna göre uçnoktaya genellikle sorguların bu küçük kısmı yollanmakta-

dır. BSBM kütüphanesiyle oluşturulan otomatik veriler, 100M üçlü içerecek şekilde 4store'a yüklenmiştir ve bir sparql uçnokta ile sorgulanmaya açılmıştır. Daha sonra uçnoktaya gelen sorguların aynı sorgu olma olasılıklarıyla, pareto dağılımındaki α değerinin değeri 0.1 - 4 (%64 - %90 aynı sorgunun atılma ihtimali) arasında oynanarak, keşif kullanım durumu sorguları uçnokta üzerinde çalıştırılmış ve performans ölçümleri yapılmıştır. Değerlendirme yapılırken güncellemeler dikkate alınmamış, veri kümesindeki verilerin değişmez olduğu varsayılmıştır.

6.1.2 E-Ticaret keşif kullanım durumu

Bellek Çizgesi, genişletilen BSBM içerisinde uygulama ile uçnokta katmanları arasına tümleştirilmiştir. α değerleri sırasıyla 0.1 - 0.25 - 0.5 - 1 - 2 - 3 - 4 arasında değiştirilerek performans ölçümleri gerçekleştirilmiştir. Keşif kullanım durumundaki 12 sorgu taslağından oluşan 25 sorguluk sorgu karışımı her α değeri için 100er kez çalıştırılmış, asıl sorgular çalıştırılmadan önce 10ar sorgu karışımı ısıtma olarak çalıştırılmış ve zaman aşım süresi 90 saniye olarak belirlenmiştir. Isıtma için çalıştırılan sorgular çıkarıldığında 2500 sorgu her α değeri için hem Bellek Çizgesi kullanılarak, hem de kullanılmayarak çalıştırılmıştır. Uçnokta performansının değerlendirildiği ölçütler BSBM tarafından şu şekilde belirlenmiştir:

Tekil Sorgular İçin Ölçütler

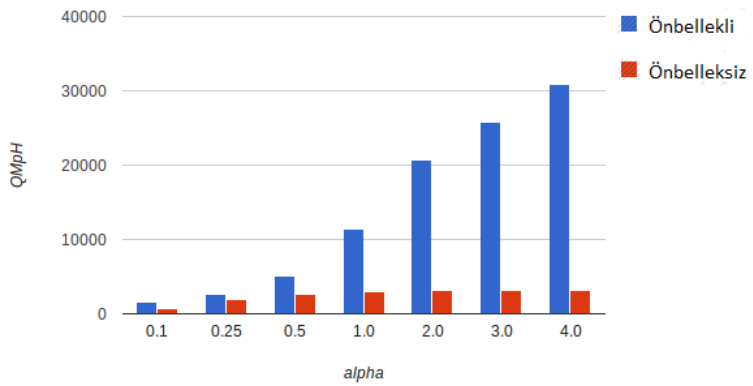
- **Ortalama Sorgu İşletim Süresi (aQET):** Bir sorgu taslağının sınanan sistemde, farklı parametreler ile ortalama çalışma süresini ifade eder.

- **Saniye Başına Düşen Sorgu (QpS):** Bir sorgu taslağının saniyede kaç defa çalıştırabileceğini tanımlar.

- **En Küçük/En Büyük Sorgu İşletim Süresi (minQET, maxQET):** Bir sorgu taslağının en düşük ve en yüksek çalıştırma süresidir.

Sorgu Karışımları İçin Ölçüler

- **Saat Başına Düşen Sorgu Karışımı (QMpH):** Sınanan sistemde 1 saatte farklı parametreler ile çalıştırılabilecek sorgu karışımı sayısını tanımlar.

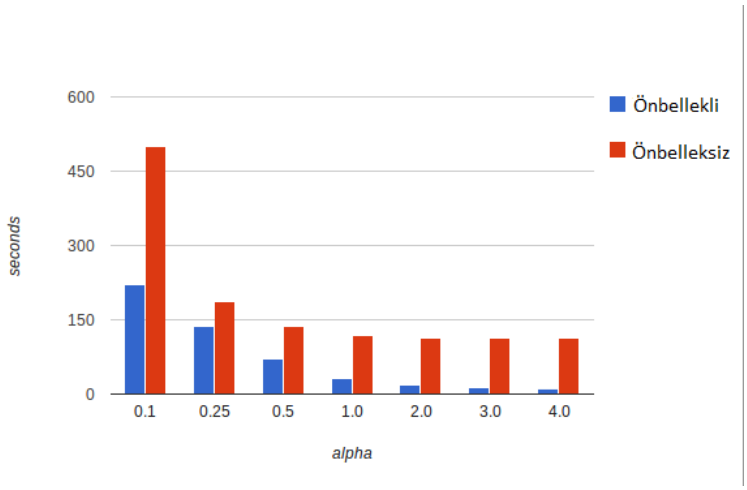


Şekil 6.1. QMpH'ın α'ya Göre Değişimi

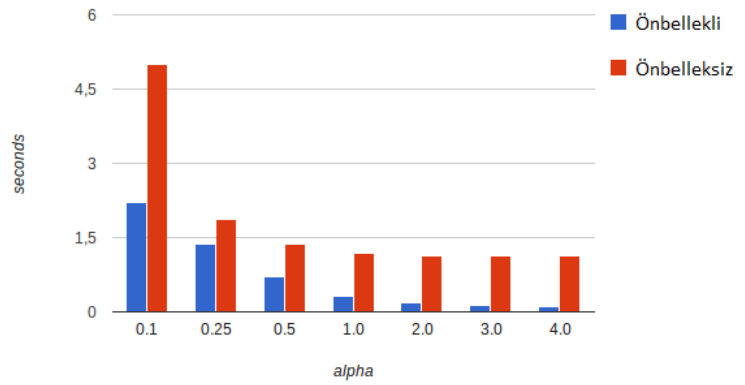
- **Toplam Çalışma Süresi (oaRT):** Test sürücüsünün sınanan sistem üzerinde çalıştırdığı belirli sayıdaki sorgu karışımı için geçen toplam süredir.
- **Birleşik Sorgu İşletim Süresi (cQET):** Bir sorgu karışımının birçok kez farklı parametreler ile çalıştırılma süresidir.
- **Tüm Sorgular Üzerinde Ortalama Sorgu İşletim Süresi (aQEToA):** 50 sorgu karışımını çalıştırmak için geçen süresinin toplam sorgu sayısına bölünmesini ifade eder.

Tanımlanan bu ölçütler içerisinden en önemli olanları ise Saniye Başına Düşen Sorgu (QpS), Saat Başına Düşen Sorgu Karışımı (QMpH) ve Toplam Çalışma Süresi (oaRT)'dir.

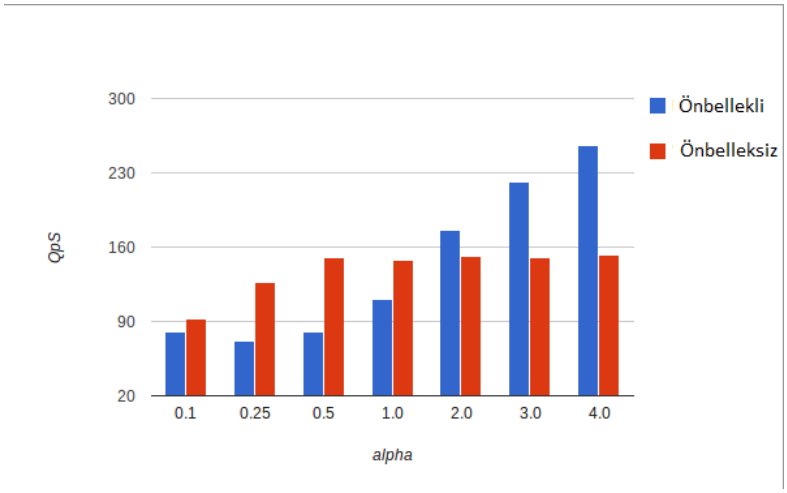
Saat başına düşen sorgu karışımı (QMpH) değerleri Şekil 6.1'de gösterilmektedir. Buna göre Bellek Çizgesi yaklaşımı ile çok daha fazla sorgu işletilebilmektedir. Sorgunun toplam çalışma süresi (oaRT) değerlerinin temsil edildiği Şekil 6.2'de görüldüğü üzere, kullanım durumundaki sorgu karışımı, Bellek Çizgesi ile oldukça kısa sürede çalıştırılabilir. Sorgu karışımının farklı parametreler ile çalıştırma süreleri Şekil 6.3'te gösterilmiştir ve buna göre Bellek Çizgesi yaklaşımı ile sorgu karışımının çok daha hızlı çalıştırıldığı görülmektedir. Şekil 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10, 6.11, 6.12, 6.13 ve 6.14'te 12 sorgu taslağından 11'inin saniye başına düşen sorgu değerleri verilmiştir. Bu grafikler sorguların saniyede kaç kere işletilebildiğini temsil etmektedir. 6. sorgu taslağı için çalıştırma sonucunda herhangi bir sonuç elde edilmediğinden, QpS bakımından grafiksel olarak temsil edilmemiştir.



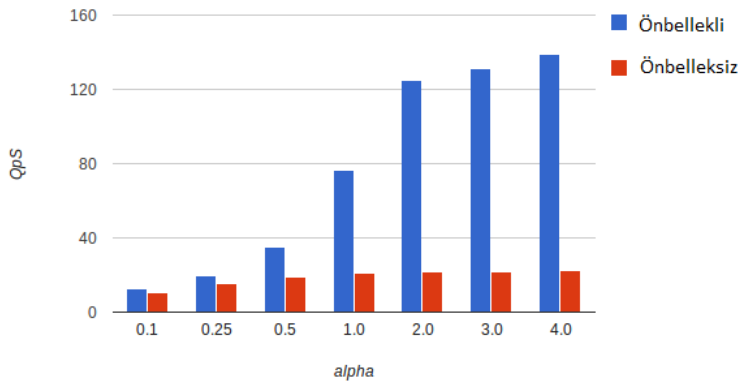
Şekil 6.2. oaRT'nin α'ya Göre Değişimi



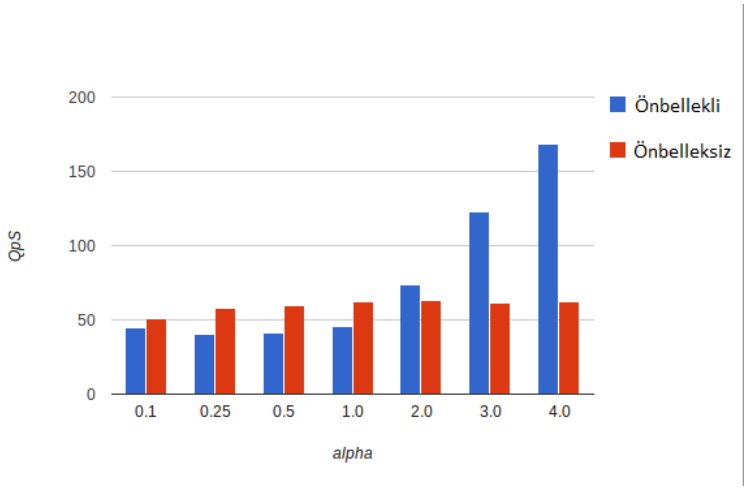
Şekil 6.3. cQET'nin α'ya Göre Değişimi



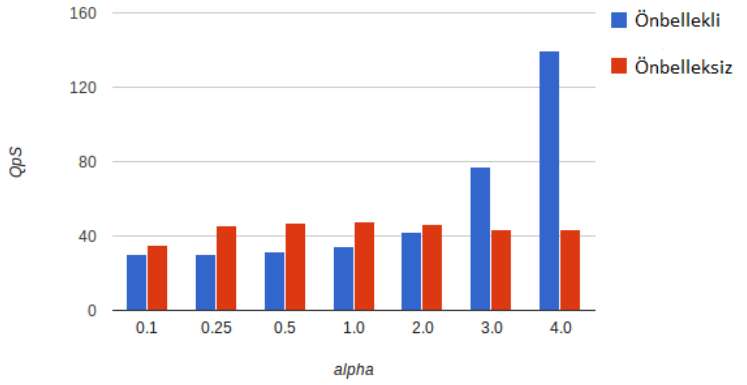
Şekil 6.4. Sorgu Taslağı-1 İçin QpS'nin α'ya Göre Değişimi



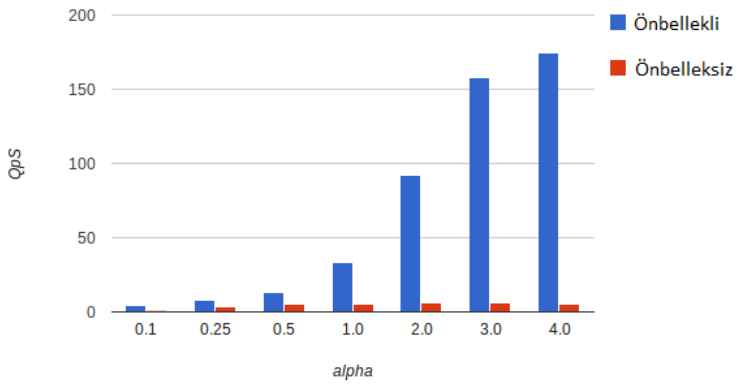
Şekil 6.5. Sorgu Taslağı-2 İçin QpS'nin α'ya Göre Değişimi



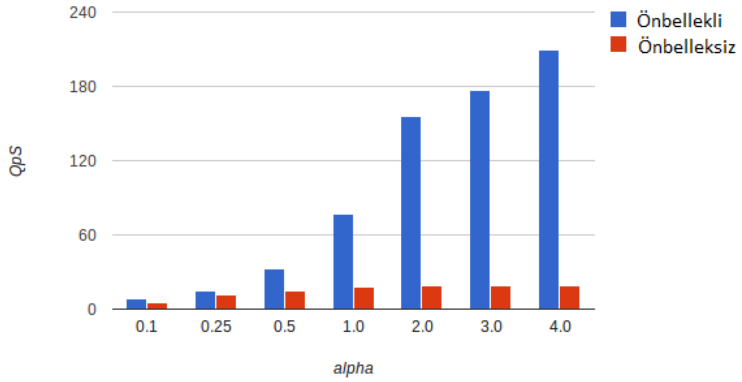
Şekil 6.6. Sorgu Taslağı-3 İçin QpS'nin α'ya Göre Değişimi



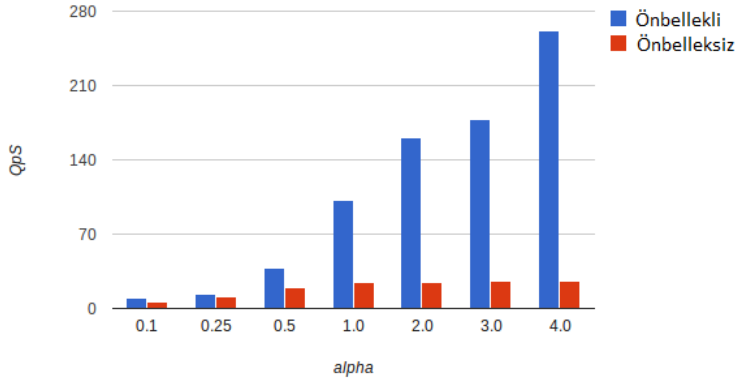
Şekil 6.7. Sorgu Taslağı-4 İçin QpS'nin α'ya Göre Değişimi



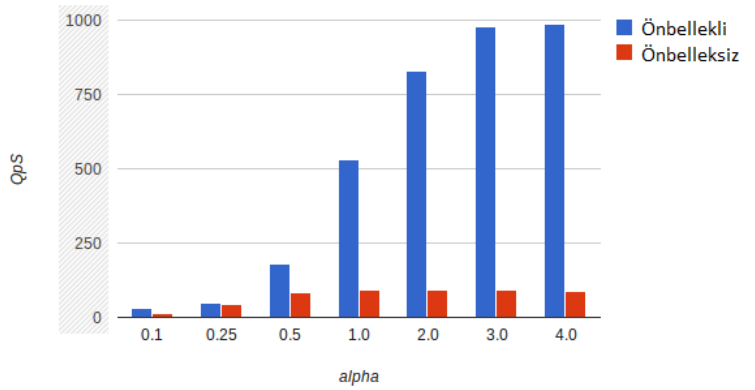
Şekil 6.8. Sorgu Taslağı-5 İçin QpS'nin α'ya Göre Değişimi



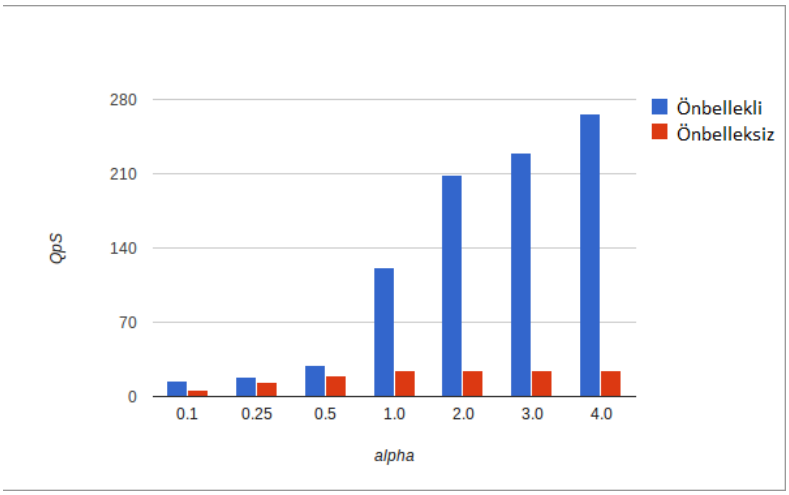
Şekil 6.9. Sorgu Taslağı-7 İçin QpS'nin α'ya Göre Değişimi



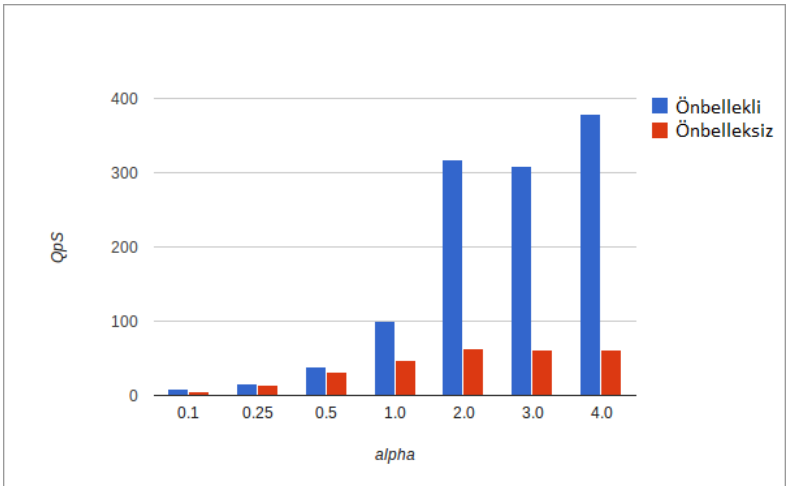
Şekil 6.10. Sorgu Taslağı-8 İçin QpS'nin α'ya Göre Değişimi



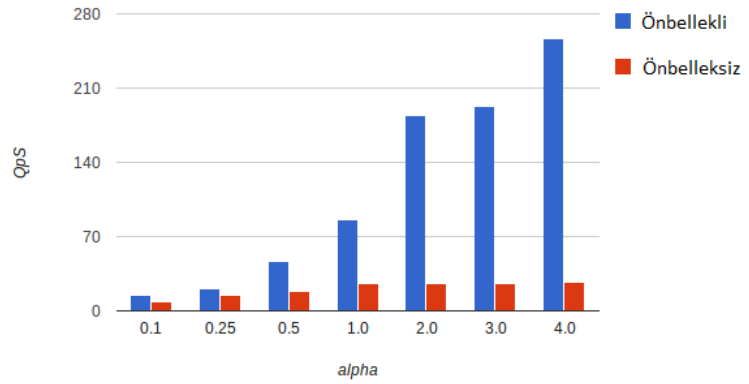
Şekil 6.11. Sorgu Taslağı-9 İçin QpS'nin α'ya Göre Değişimi



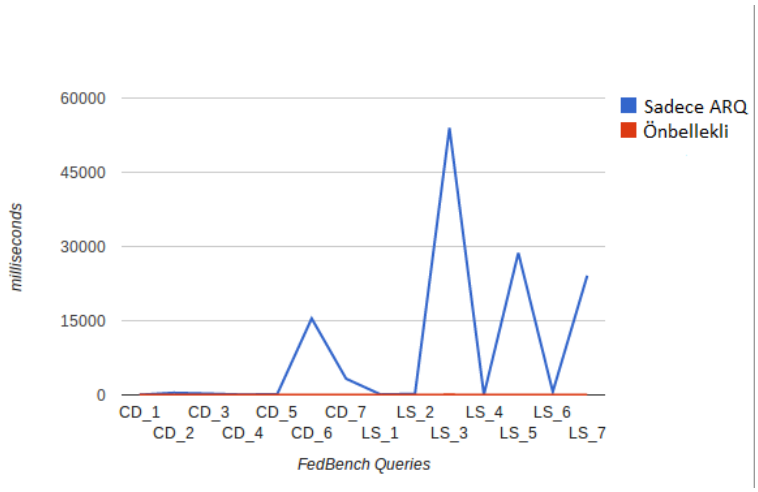
Şekil 6.12. Sorgu Taslağı-10 İçin QpS'nin α'ya Göre Değişimi



Şekil 6.13. Sorgu Taslağı-11 İçin QpS'nin α'ya Göre Değişimi



Şekil 6.14. Sorgu Taslağı-12 İçin QpS'nin α'ya Göre Değişimi



Şekil 6.15. ARQ-SPARQL Önbellek FedBench Sorguları Çalıştırma Karşılaştırması

6.2 Birleştirilmiş Sorgularda Değerlendirme

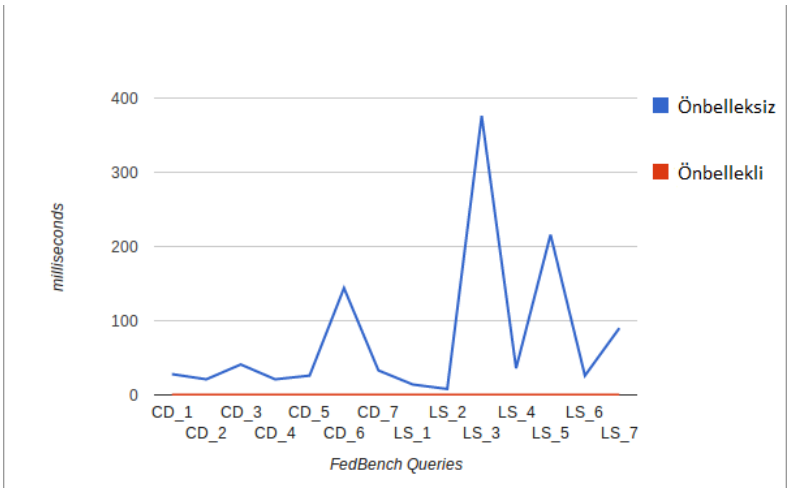
Bu bölümde FedBench sorgularını oluşturan 7 tane Çapraz Alan ve 7 tane de Hayat Bilimleri, sorgusunun birleştirilmiş halleri önce sadece ARQ ile, daha sonra da Bellek Çizgesi ile çalıştırılmıştır. İki durumdaki sorgu çalıştırma süreleri tüm sorgu sonuçları elde edilecek şekilde hesaplanmıştır. Bu bölümde vurgulanmak istenen asıl nokta, sadece ARQ kullanılarak bu birleştirilmiş sorguları çalıştırmanın ne kadar maliyetli olduğu ve uygulanan Bellek Çizgesi yönteminin bu maliyetin üzerinden kolayca nasıl gelebildiğidir. Şekil 6.15'te FedBench kıyaslama kütüphanesi sorgularının yalnız ARQ ve Bellek Çizgesi ile çalıştırma sonuçları görülmektedir. Sonuçlar ortaya konurken ilk iki sorgu ısırtma olarak çalıştırılmıştır. Grafikten anlaşılacağı üzere ARQ ile geçen çalıştırma süresi, Bellek Çizgesi kullanıldığında ortadan kalkmaktadır.

6.3 SPARQL Önbelleklemenin WoDQA'ya Uygulanması

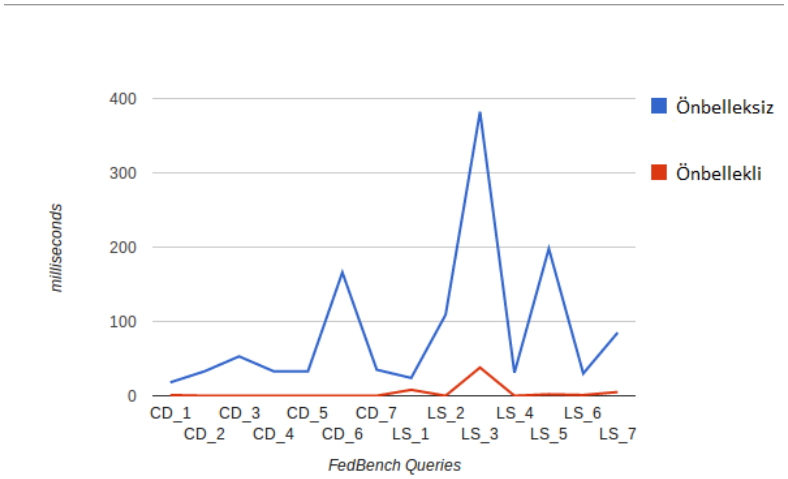
Önbellekleme değerlendirmesinin son bölümünde Bellek Çizgesi, WoDQA'ya tümleştirilmiş ve WoDQA'nın performansına olan etkileri ortaya konmuştur. İlk bölümde Berlin SPARQL Kıyaslama Kütüphanesi kullanılarak yöntemin tek uçnoktaya atılan sorgulardaki etkisi saat başına düşen sorgu karışımı (QMpH), toplam çalışma süresi (oaRT), birleşik sorgu işletim süresi (cQET) ve saniye başına düşen sorgu (QpS) bakımından detaylıca incelenmiştir ve gözle görülür etkisi ispatlanmıştır. Birleştirilmiş sorgulama ve tek uçnoktalı sorgulama sadece işletim yönüyle birbirinden farklıdır ve bu iki tipteki sorguların da Bellek Çizgesi'nde saklanması ve önbellekteki sonuçların elde edilmesi arasında hiçbir fark yoktur. Bu yüzden birleştirilmiş sorguların QMpH, oaRT, cQET ve QpS gibi ölçütler yönünden değerlendirilmesi ayrıca yapılmamıştır, çünkü sonuçların yapılan tekil uçnoktalı sorgu değerlendirmesiyle eşdeğer olacağı açıktır.

WoDQA'nın Bellek Çizgesi ile tümleşimsiz ve tümleşimli çalıştırma sonuçları, ilk sonuçlar ve tüm sonuçlar elde edilerek, iki şekilde de ortaya konmuştur. İlk sonucu elde ederek çalıştırma süreleri Şekil 6.16'da, tüm sonuçları elde ederek çalıştırma süreleri ise Şekil 6.17'de ortaya konmuştur. Sonuçlar elde edilirken ilk iki sorgu ısıtma olarak çalıştırılmıştır. Grafiklerin de açıkça ortaya koyduğu gibi WoDQA sorguları önbelleksiz çok hızlı çalıştırıyor olsa bile, önbellek kullanıldığında çok daha üstün bir tepki süresine erişmektedir. Şekil 6.16'da ilk sonuçların elde edildiği değerlendirmede, sonuçların dolaşılması hesaba katılmayıp, sadece sonuçları elde etme süresi gözönüne alındığından tepki süresinin sıfıra indiği, Şekil 6.17'de de benzer olarak tüm sonuçların dolaşıldığı durumda tepki süresinin sıfıra yakınsadığı gözlemlenmiştir.

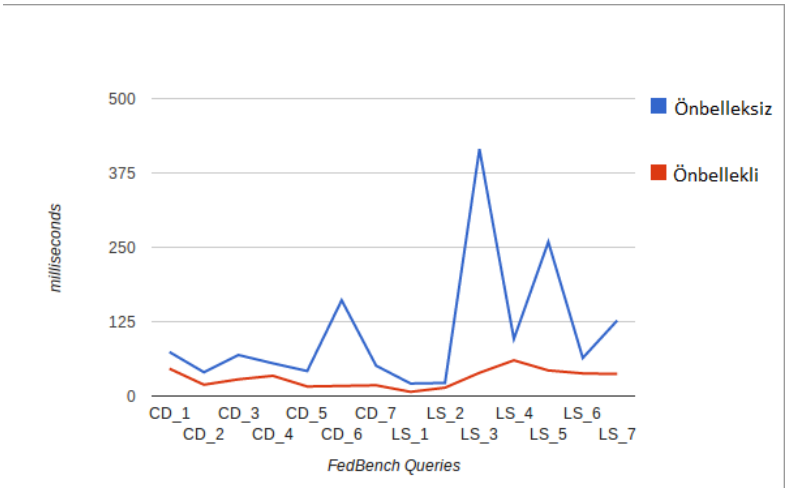
Bunlara ek olarak Bellek Çizgesi'nin WoDQA'nın tüm performansına olan etkisi (çözümleme+çalıştırma), Şekil 6.18'de ilk sonuç elde edilerek ve Şekil 6.19'da tüm sonuçlar elde edilerek temsil edilmiştir. SPARQL sorgularının ham olarak alınıp birleştirilmiş sorguya dönüştürülmesi ve oluşturulan birleştirilmiş sorguların çalıştırılması sorgu değerlendirme olarak ifade edilir. Şekil 6.18 ve Şekil 6.19'dan görüleceği gibi, Bellek Çizgesi ile WoDQA'nın performansı eniyileştirilerek, sorgu değerlendirme hızı oldukça artırılmıştır.



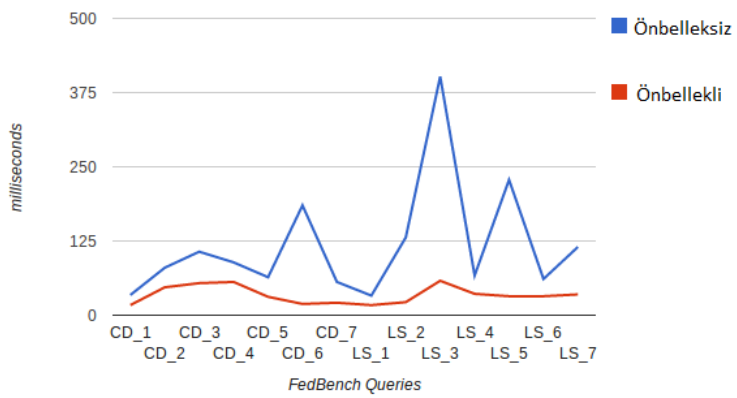
Şekil 6.16. WoDQA ile önbellek(li/siz) Sorgu Çalıştırma (İlk Sonuç)



Şekil 6.17. WoDQA ile önbellek(li/siz) Sorgu Çalıştırma (Tüm Sonuçlar)



Şekil 6.18. WoDQA ile önbellek(li/siz) Sorgu Değerlendirme (İlk Sonuç)



Şekil 6.19. WoDQA ile önbellek(li/siz) Sorgu Değerlendirme (Tüm Sonuçlar)

7 SONUÇ

Bu tezde, VOID veri kümesi üstverisini kullanarak bir sorgunun ilişkili veri kümelerini bulan ve sorguyu bu ilişkili veri kümeleri üzerine dağıtan WoDQA tanıtılmış ve WoDQA'da gerçekleştirilen performans eniyileştirmelerinden bahsedilmiştir. WoDQA, bir veri kümesinin içeriğini ve bağlı veri bulutunun topolojisini belirlemek için VOID'in URI uzayı, sözlük ve bağ kümesi kavramlarını kullanır. Bu tanımlayıcı özellikler bağlı veri bulutunun kullanışlı bir özetini sunar. VOID'in istatistiksel özelliklerinden yararlanılmamıştır, çünkü açık ve değişken bağlı veri bulutlarında tanımlayıcı özelliklerin güncelliğini yönetmek istatistiksel olanlarınkini yönetmekten daha kolaydır. WoDQA sorgu birleştirmenin 3 adımını gerçekleştirmektedir. İlk adımda, WoDQA her üçlü deseni için ilişkili veri kümelerini çözümlemesinin yanısıra üçlü deseni ilişkilerini ve verikümeleri arasındaki bağları da çözümlemeye dahil eden kapsamlı bir veri kümesi seçimi mekanizmasını içerir. İkinci adımda WoDQA, harici grupları oluşturarak ve bazı sezgisel yöntemlerle sorgu çalıştırma düzenini belirleyerek sorgu çalıştırma planını eniyileştirir. Son adımda, WoDQA sorguyu dağıtık veri kümeleri üzerinde işletirken FILTER tabanlı bağlı birleştirme yöntemini uygular.

FedBench üzerinde gerçekleştirilen çalışmadan gözlemlendiği üzere, VOID'de yer alan veri kümesi seçilimine URI uzayı, sözlük ve bağ kümesi tanımlarını kullanmak oldukça yararlı olmaktadır. Gelişmiş diğer iki birleştirilmiş sorgu motoru olan FedX ve SPLENDID ile kıyaslandığında WoDQA en iyi veri kümesi seçimi, istek sayısı ve sorgu çalıştırma süresi performansını vermiştir. Birleştirme stratejileri istek sayısını önemli ölçüde azaltarak sorgu işletim süresini düşürmektedir. Ancak üçlü depolarının, HTTP isteği sayısını azaltan bağlı birleştirme stratejileri için eniyileştirilmeleri gerekmektedir. Örneğin, yeni bir SPARQL standardı olan VALUES ifadesi bağlı birleştirmeyi uygulamak için oldukça uygundur, fakat bu ifadelerin RDF sunucuları tarafından düzgün ve etkin şekilde ele alınıp işlenmeleri gerekmektedir.

Bunların dışında gerçekleştirilen önbellekleme yöntemi ile SPARQL sorgularının etkin bir şekilde önbelleklemesi sağlanmıştır. Uygulamalarda SPARQL sorguları genellikle otomatize bir şekilde atılmaktadır, bu durumda çoğunlukla aynı sorgular tekrar sorgulanmak durumunda kalmaktadır. Bu nedenle bağlı veri uygulamalarında etkin bir önbellek mekanizması ile sorgulama maliyetinin önüne önemli ölçüde geçilebilmektedir. İşte bu nedenle gerçekleştirilen önbellekleme yöntemi ile hem tek uçnokta

	FedX	SPLENDID	WoDQA
CD-1	15	110	63
CD-2	330	80	49
CD-3	109	103	88
CD-4	100	85	72
CD-5	97	101	79
CD-6	281	10000	172
CD-7	324	3000	42
LS-1	47	200	20
LS-2	16	400	24
LS-3	1470	20000	399
LS-4	1	800	51
LS-5	480	21000	217
LS-6	34	1000	44
LS-7	481	20000	183

Tablo 7.1. Sorgu İşletim Süreleri

üzerinde çalışan sorgular hem de birleştirilmiş sorgular tekrar tekrar sorgulandığında oluşan çalıştırma maliyetinden kurtulunmuş olunur. Gerçekleştirilen önbellek mimarisi sorguları hızlı çalıştırmanın yanı sıra, üçlü önbelleklemesi ve düğüm önbelleklemesi ile bellekte her nesnenin tek bir varlığının kullanılmasını sağlamakta, böylece önbelleğin tek bir çizge şeklinde genişlemesini ve etkin şekilde yönetilmesini sağlamış olmaktadır.

Bunların dışında WoDQA'da ve önbellekleme mimarisinde eksik kalan ve geliştirilmesi gereken bazı noktalar mevcuttur. Bunlarda ilki önbellek güncelliğinin değişken bir şekilde yönetilmesidir. Bu sayede az güncellenen nesneler daha uzun sürede önbellekte yer alacakken, sık güncellenen verilerin önbellekte kalma süresi daha az olacaktır (Alıcı vd., 2012). İkincisi de önbellekte tutulacak sorgu sonuçlarını maliyete göre yönetmek olabilir (Özcan vd., 2011), maliyetli sorgu sonuçları önbellekte tutulurken, daha az maliyetli olanlar çıkarılacaktır. Üçüncü olarak uygulanan eniyileştirmeler FedBench sorgularında düzgün şekilde çalışmasına rağmen karmaşık sorgularda ve OPTIONAL üçlü desenlerinde bazı kısıtlama ve eksiklikler olduğu gözlemlenmiştir. Sorgu eniyileştirmesinin (Montoya et al., 2012b) tarafından ortaya konan karmaşık sorgular üzerinde de gerçekleştirilmesi yararlı olacaktır. Dördüncü geliştirme bağlı birleştirme yöntemi için yapılabilir. Bağlı birleştirme yöntemi blok tabanlı yineleyici kullandığından büyük veri kümeleri üzerinde sıkıntı ortaya çıkabilmektedir. Uyarlanabilir sorgu motorları (Acosta et al., 2011) ve paralelleştirmenin (Wang et al., 2013) dahil edilmesi bu durumların üstesinden gelebilmektedir. Diğer yandan benzer içeriğe sahip çok sayıda veri kümesinin bulunduğu bağlı veri bulutlarında veri kümesi seçimi başarılı olamayabilmektedir. Harici grupların paralel olarak işletilmesi bu durumlar için bir iyileştirme olabilir. Son olarak eş-referans çözümlemesi, örneğin veri kümeleri arasındaki `owl:sameAs` ilişkilerini işleme, şu anki gerçekleştirimde açık bir sorun olarak durmaktadır. Birleştirilmiş sorgu motorlarının, geliştiriciyi sorgunun içine `owl:sameAs` ilişkilerini gömmekten koruması gerekmektedir. Bağ yüklemi `owl:sameAs` olan bağ kümesi tanımlamaları sorgu motorunun eş-referans çözümlemesini geliştirmesine yardımcı olabilmektedir.

KAYNAKLAR DİZİNİ

Acosta, M., Vidal, M.-E., Lampo, T., Castillo, J., and Ruckhaus, E., 2011, Anapsid:an adaptive query processing engine for sparql endpoints, In Proceedings of the 10th international conference on The semantic web - Volume Part I, ISWC'11, Berlin, Heidelberg. Springer-Verlag, 18-34pp.

Akar,Z., Halaç, T. G., Ekinci, E. E. ve Dikenelli, O., 2012, Querying the web of interlinked datasets using void descriptions, In Bizer, C., Heath, T., Berners-Lee, T., and Hausenblas, M., editors, LDOW, volume 937 of CEUR Workshop Proceedings. CEUR-WS.org.

Alexander, K., Cyganiak, R., Hausenblas, M., and Zhao, J., 2009, Describing linked datasets - on the design and usage of voiD, the 'Vocabulary of Interlinked Datasets', In WWW 2009 Workshop: Linked Data on the Web (LDOW2009), Madrid, Spain.

Alıcı, S., Altıngövde, I. S., Özcan, R., Cambazoğlu, B. B. Ve Ulusoy, Ö., 2012, Adaptive time-to-live strategies for query result caching in web search engines, In Baeza-Yates, R. A., de Vries, A. P., Zaragoza, H., Cambazoğlu, B. B., Murdock, V., Lempel, R., and Silvestri, F., editors, ECIR, volume 7224 of Lecture Notes in Computer Science, Springer, 401-412pp.

Bernstein, A., Kiefer, C., and Stocker, M., 2007, OptARQ: A SPARQL Optimization Approach based on Triple Pattern Selectivity Estimation, Technical Reportifi-2007.03, Department of Informatics, University of Zurich.

Bizer, C., Heath, T., Ayers, D., and Raimond, Y., 2007, Interlinking open data on the web, www4.wiwiss.fu-berlin.de/bizer/pub/LinkingOpenData.pdf, Stand 12.5.2009

Bizer, C. and Schultz, A., 2009, The Berlin SPARQL Benchmark, Int. J. Semantic Web Inf. Syst., 5(2):1–24pp.

Bouquet, P., Ghidini, C., and Serafini, L., 2009, Querying the web of data: A formal approach, In Proceedings of the 4th Asian Conference on The Semantic Web, ASWC'09, Berlin, Heidelberg. Springer-Verlag, 291–305pp.

Buil, C., Arenas, M., and Corcho, O., 2011, Semantics and optimization of the SPARQL 1.1 Federation Extension, In Proc. of 8th Extended Semantic

KAYNAKLAR DİZİNİ (devam)

Web Conference (ESWC 2011), volume 6644 of Lecture Notes in Computer Science, Heraklion,Crete,Greece, 1-15pp.

Duerst, M. and Suignard, M., 2005, Internationalized Resource Identifiers (IRIs), RFC 3987 (Proposed Standart).

Görlitz, O. and Staab, S., 2011, Federated Data Management and Query Optimization for Linked Open Data, In Vakali, A. and Jain, L., editors, New Directions in Web Data Management 1, volume 331 of Studies in Computational Intelligence, Springer Berlin/Heidelberg, 109–137pp.

Görlitz, O. and Staab, S., 2011, SPLENDID: Sparql endpoint federation exploiting void descriptions, In Proceedings of the 2nd International Workshop on Consuming Linked Data, Bonn, Germany.

Haase, P., Mathäß, T., and Ziller, M., 2010, An evaluation of approaches to federated query processing over linked data, In Proceedings of the 6th International Conference on Semantic Systems, New York, NY, USA.

Halaç, T., Yönyül, B., Akar, Z., ve Dikenelli, O., 2013, WoDQA: VoID Based Query Federation Engine for Linked Data (under review), Web semantics science, services and agents on the World Wide Web.

Hartig, O., 2011, Zero-knowledge query planning for an iterator implementation of link traversal based query execution, In Antoniou, G., Grobelnik, M., Simperl, E. P. B., Parsia, B., Plexousakis, D., Leenheer, P. D., and Pan, J., editors, ESWC (1), volume 6643 of Lecture Notes in Computer Science, Springer, 154-169pp.

Hartig, O., Bizer, C., and Freytag, J. C., 2009, Executing sparql queries over the web of linked data, In International Semantic Web Conference, 293-309pp.

Hartig, O. and Langegger, A., 2010, A Database Perspective on Consuming Linked Data on the Web, Datenbankspektrum, Semantic Web Special Issue.

Heath, T. and Bizer, C., 2011, Linked Data: Evolving the Web into a Global Data Space, Morgan & Claypool, San Rafael, CA, 1 edition.

Johannes Lorey, F. N, 2013, Caching and prefetching strategies for sparql queries, In ESWC 2013 Satellite Events – Revised Selected Papers, Montpellier, France.

KAYNAKLAR DİZİNİ (devam)

Martin, M., Unbehauen, J., and Auer, S., 2010, Improving the performance of semantic web applications with sparql query caching, In Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., and Tudorache, T., editors, Proceedings of 7th Extended Semantic Web Conference (ESWC 2010), 30 May – 3 June 2010, Heraklion, Crete, Greece, volume 6089 of Lecture Notes in Computer Science, Berlin / Heidelberg. Springer, 304-318pp.

Montoya, G., Vidal, M.-E., and Acosta, M., 2012, A heuristic-based approach for planning federated sparql queries, In COLD.

Montoya, G., Vidal, M.-E., Corcho, O., Ruckhaus, E., and Buil-Aranda, C., 2012, Benchmarking federated sparql query engines: are existing testbeds enough?, In Proceedings of the 11th international conference on The Semantic Web – Volume Part II, ISWC'12, Berlin, Heidelberg. Springer-Verlag, 313-324pp.

Özcan, R., Altıngövde, I. S., ve Ulusoy, Ö., 2011, Cost-Aware Strategies for Query Result Caching in Web Search Engines, TWEB, 5(2):9.

Quilitz, B. and Leser, U., 2008, Querying Distributed RDF Data Sources with SPARQL, In Bechhofer, S., Hauswirth, M., Hoffmann, J., and Koubarakis, M., editors, The Semantic Web: Research and Applications, volume 5021 of Lecture Notes in Computer Science, chapter 39, Springer Berlin / Heidelberg, Berlin, Heidelberg, 524–538pp.

Rietveld, L., 2012, Replication for linked data, In Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J. X., Hendler, J., Schreiber, G., Bernstein, A., and Blomqvist, E., editors, International Semantic Web Conference (2), volume 7650 of Lecture Notes in Computer Science, Springer, 415–423pp.

Schandl, B., 2010, Replication and versioning of partial rdf graphs, In Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., and Tudorache, T., editors, ESWC (1), volume 6088 of Lecture Notes in Computer Science, Springer, 31-45pp.

Schwarte, A., Haase, P., Hose, K., Schenkel, R., and Schmidt, M., 2011, Fedx: A Federation Layer for Distributed Query Processing on Linked Open Data, In Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., and Pan, J., editors, The Semanic Web:

KAYNAKLAR DİZİNİ (devam)

Research and Applications, volume 6644 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 481-486pp.

Schwarte, A., Haase, P., Hose, K., Schenkel, R., and Schmidt, M., 2011, Fedx: Optimization Techniques for Federated Query Processing on Linked Data, In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., and Blomqvist, E., editors, The Semantic Web - ISWC 2011, volume 7031 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 601-616pp.

Wang, X., Tiropanis, T., and Davis, H. C., 2013, Lhd: Optimising linked data query processing using parallelisation, In LDOW.

Williams, G. T. and Weaver, J., 2011, Enabling fine-grained http caching of sparql query results, In Aroyo, L.,Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N. F., and Blomqvist, E., editors, International Semantic Web Conference (1), volume 7031 of Lecture Notes in Computer Science, Springer, 762-777pp.

ÖZGEÇMİŞ

Ad Soyad	Burak Yönyül
Doğum Tarihi	18.08.1988
Doğum Yeri	İzmir
Uyruğu	T.C.
Eğitim	
Yüksek Lisans	2011-... Ege Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı
Lisans	2006-2011 Ege Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü
Lise	2002-2006 İzmir Kız Lisesi
Araştırma Alanları	Anlamsal veb, bağlı veri, birleştirilmiş sorgulama önbellekleme, SPARQL

EKLER

Ek 1 Türkçe–İngilizce Terimler Sözlüğü

Ek 2 Birleştirilmiş Sorgular

Ek 3 FedBench Yerel Veri Kümesi Uçnokta Adresleri

Ek1 Türkçe - İngilizce Terimler Sözlüğü

Alt sorgu: Subquery

Anlamsal: Semantic

Bağ: Link

Bağlı açık veri bulutu: Linked open data cloud

Bağ dolanımı: Link traversal

Bağlı Birleştirme: Bound Join

Bağlı Veri: Linked Data

Bağ Kümesi: Linkset

Bellek Çizgesi: Memory Graph

Birleşik Sorgu İşletim Süresi: Composite Query Execution Time

Birleştirme: Join

Birleştirilmiş sorgu: Federated query

Biçimselleştirme: Formalization

Çalıştırma: Execution

Çizge: Graph

Çözümleme süreci: Anaylsis process

Dağıtık: Distributed

Değerlendirme: Evaluation

Değişken: Variable

Dizin: İndex

Dolaşıcı: Iterator

Döngüsel Yuvalanmış Birleştirme: Nestel Loop Join

Düğüm: Node

Düğüm Önbelleği: Node Cache

Eniyileştirme: Optimization

En Küçük/En Büyük Sorgu İşletim Süresi: Min/Max Query Execution Time

Ham sorgu: Raw query

Harici Grup: Exclusive Group

Hash şifresi: Hashcode

Hedef veri kümesi: Referenced dataset

Hizmetin Kalitesi: Quality of Service

İlişkili veri kümesi: Relevant dataset

İlişkisiz veri kümesi: Irrelevant dataset

İşletme: Processing

Kaynak: Resource

Kaynak veri kümesi: Referrer dataset

Kıyaslama Kütüphanesi: Benchmark

Kullanım Durumu: Use Case

Nesne: Object

Ortalama Sorgu İşletim Süresi : Average Query Execution Time

Ödün verme: Trade-off

Önek: Prefix

Önden getirme: Prefetching

Ön işleme: Preprocessing

Özne: Subject

Önbellek: Cache

Önbellek bakımı: Cache maintenance

Önbellek yönetimi: Cache management

Parçacıklı: Granularity

Saat Başına Düşen Sorgu Karışımı: Queries Mixes per Hour

Saniye Başına Düşen Sorgu: Queries per Second

Servis çizge deseni: Service graph pattern

Sezgisel: Heuristic

Sistem Modeli: System Model

Sorgu Çalıştırıcısı: Query Executor

Sorgu Düzenleyici: Query Reorganizer

Sorgu federasyonu: Query federation

Sorgulayıcı Birimi: Querier Module

Sorgu Önbelleği: Query Cache

Sorgu Yapılandırıcı Birimi: Query Builder Module

Söz dizimi: Syntax

Sözlük : vocabulary

Tahliye Birimi: Eviction Module

Tamlık: Completeness

Tek tipleştirme: Standardization

Toplam Çalışma Süresi: Overall Runtime

Tüm Sorgular Üzerinde Ortalama Sorgu İşletim Süresi: Average Query Execution Time over all Queries

Uçnokta: Endpoint

Üçlü: Triple

Üçlü deseni: Triple pattern

Üçlü Önbelleği: Triple CAche

Üstveri: Metadata

Veri ambarlama: Datawarehousing

Veri deposu: Data store

Veri kümesi: Dataset

Veri Kümesi Çözümleyicisi: Dataset Analyzer

Veri Kümesi Keşif Kuralları: Dataset Discovery Rules

VOID deposu: VOID store

Yayımlayıcı: Publisher

Yüklem: Predicate

Ek2 Birleştirilmiş Sorgular

Birleştirilmiş Çapraz Alan Sorgusu-1:

```
SELECT  ?predicate ?object
WHERE
  {   { SERVICE <http://localhost:7000/sparql/>
          { dbpedia:Barack_Obama ?predicate ?object }
      }
    UNION
      { SERVICE <http://localhost:9000/sparql/>
          { ?subject owl:sameAs dbpedia:Barack_Obama .
            ?subject ?predicate ?object
          }
      }
  }
```

Birleştirilmiş Çapraz Alan Sorgusu-2:

```
SELECT  ?party ?page
WHERE
  { SERVICE <http://localhost:7000/sparql/>
      { dbpedia:Barack_Obama dbpont:party ?party }
    SERVICE <http://localhost:9000/sparql/>
      { ?x owl:sameAs dbpedia:Barack_Obama .
        ?x nytimes:topicPage ?page
      }
  }
```

Birleştirilmiş Çapraz Alan Sorgusu-3:

```
SELECT  ?president ?party ?page
WHERE
  { SERVICE <http://localhost:7000/sparql/>
      { ?president rdf:type dbpont:President .
        ?president dbpont:nationality dbpedia:United_States .
        ?president dbpont:party ?party
      }
    SERVICE <http://localhost:9000/sparql/>
      { ?x owl:sameAs ?president .
        ?x nytimes:topicPage ?page
      }
  }
```

Birleştirilmiş Çapraz Alan Sorgusu-4:

```
SELECT  ?actor ?news
WHERE
  { SERVICE <http://localhost:2500/sparql/>
      { ?film dc:title "Tarzan" .
        ?film lmdbvoc:actor ?actor .
        ?actor owl:sameAs ?x
      }
    SERVICE <http://localhost:9000/sparql/>
      { ?y nytimes:topicPage ?news .
        ?y owl:sameAs ?x
      }
  }
```

Birleştirilmiş Çapraz Alan Sorgusu-5:

```
SELECT  ?film ?director ?genre
WHERE
  { SERVICE <http://localhost:7000/sparql/>
      { ?director dbpont:nationality dbpedia:Italy .
        ?film dbpont:director ?director
      }
    SERVICE <http://localhost:2500/sparql/>
      { ?x owl:sameAs ?film .
        ?x lmdbvoc:genre ?genre
      }
  }
```

Birleştirilmiş Çapraz Alan Sorgusu-6:

```
SELECT  ?name ?location ?news
WHERE
  { SERVICE <http://localhost:2000/sparql/>
      { ?germany geovoc:name "Federal Republic of Germany" .
        ?location geovoc:parentFeature ?germany
      }
      { BIND(<http://localhost:5500/sparql/> AS ?ser785260) }
    UNION
      { BIND(<http://localhost:5000/sparql/> AS ?ser785260) }
    SERVICE ?ser785260
      { ?artist foaf:based_near ?location }
      { BIND(<http://localhost:5500/sparql/> AS ?ser352667) }
    UNION
      { BIND(<http://localhost:5000/sparql/> AS ?ser352667) }
    SERVICE ?ser352667
      { ?artist foaf:name ?name }
  }
```

Birleştirilmiş Çapraz Alan Sorgusu-7:

```
SELECT  ?location ?news
WHERE
  { SERVICE <http://localhost:2000/sparql/>
      { ?parent geovoc:name "California" .
        ?location geovoc:parentFeature ?parent
      }
    SERVICE <http://localhost:9000/sparql/>
      { ?y owl:sameAs ?location .
        ?y nytimes:topicPage ?news
      }
  }
```

Birleştirilmiş Hayat Bilimleri Sorgusu-1:

```
SELECT  ?drug ?melt
WHERE
  {   { SERVICE <http://localhost:8000/sparql/>
          { ?drug drugont:meltingPoint ?melt }
      }
    UNION
      { { ?drug dbpont:Drug/meltingPoint ?melt } }
  }
```

Birleştirilmiş Hayat Bilimleri Sorgusu-2:

```
SELECT  ?predicate ?object
WHERE
  {   { SERVICE <http://localhost:8000/sparql/>
          { drugbank:DB00201 ?predicate ?object }
      }
    UNION
      { SERVICE <http://localhost:8000/sparql/>
          { drugbank:DB00201 owl:sameAs ?caff }
          { BIND(<http://localhost:8000/sparql/> AS ?ser502669) }
        UNION
          { BIND(<http://localhost:3000/sparql/> AS ?ser502669) }
        UNION
          { BIND(<http://localhost:7000/sparql/> AS ?ser502669) }
        UNION
          { BIND(<http://localhost:4000/sparql/> AS ?ser502669) }
        SERVICE ?ser502669
          { ?caff ?predicate ?object }
      }
  }
```

Birleştirilmiş Hayat Bilimleri Sorgusu-3:

```
SELECT  ?Drug ?IntDrug ?IntEffect
WHERE
  { SERVICE <http://localhost:7000/sparql/>
      { ?Drug rdf:type dbpont:Drug }
    SERVICE <http://localhost:8000/sparql/>
      { ?y owl:sameAs ?Drug .
        ?Int drugont:interactionDrug1 ?y .
        ?Int drugont:interactionDrug2 ?IntDrug .
        ?Int drugont:text ?IntEffect
      }
  }
```

Birleştirilmiş Hayat Bilimleri Sorgusu-4:

```
SELECT  ?drugDesc ?cpd ?equation
WHERE
  { SERVICE <http://localhost:8000/sparql/>
      { ?drug drugont:drugCategory drugcategory:cathartics .
        ?drug drugont:keggCompoundId ?cpd .
        ?drug drugont:description ?drugDesc
      }
    SERVICE <http://localhost:4000/sparql/>
      { ?enzyme rdf:type kegg:Enzyme .
        ?enzyme kegg:xSubstrate ?cpd .
        ?reaction kegg:xEnzyme ?enzyme .
        ?reaction kegg:equation ?equation
      }
  }
```

Birleştirilmiş Hayat Bilimleri Sorgusu-5:

```
SELECT  ?drug ?keggUrl ?chebiImage
WHERE
  { SERVICE <http://localhost:8000/sparql/>
      { ?drug rdf:type drugont:drugs .
        ?drug drugont:keggCompoundId ?keggDrug .
        ?drug drugont:genericName ?drugBankName
      }
      { BIND(<http://localhost:4000/sparql/> AS ?ser085534) }
    UNION
      { BIND(<http://localhost:3000/sparql/> AS ?ser085534) }
    SERVICE ?ser085534
      { ?keggDrug bio2rdf:url ?keggUrl }
    SERVICE <http://localhost:3000/sparql/>
      { ?chebiDrug dc:title ?drugBankName .
        ?chebiDrug bio2rdf:image ?chebiImage
      }
  }
```

Birleştirilmiş Hayat Bilimleri Sorgusu-6:

```
SELECT  ?drug ?title
WHERE
  { SERVICE <http://localhost:8000/sparql/>
      { ?drug drugont:drugCategory drugcategory:micronutrient .
        ?drug drugont:casRegistryNumber ?id
      }
    SERVICE <http://localhost:4000/sparql/>
      { ?keggDrug rdf:type kegg:Drug .
        ?keggDrug bio2rdf:xRef ?id .
        ?keggDrug dc:title ?title
      }
  }
```

Birleştirilmiş Hayat Bilimleri Sorgusu-7:

```
SELECT  ?drug ?transform ?mass
WHERE
  { SERVICE <http://localhost:8000/sparql/>
      { ?drug drugont:affectedOrganism "Humans and other mammals" .
        ?drug drugont:casRegistryNumber ?cas
        OPTIONAL
          { ?drug drugont:biotransformation ?transform }
      }
    SERVICE <http://localhost:4000/sparql/>
      { ?keggDrug bio2rdf:xRef ?cas
        { ?keggDrug bio2rdf:mass ?mass
          FILTER ( ?mass > "5" )
        }
      }
  }
```

Ek3 FedBench Yerel Veri Kümesi Uçnokta Adresleri

Veri Kümesi	Uçnokta Adresi
geonames	http://localhost:2000/sparql/
lmdb	http://localhost:2500/sparql/
chebi	http://localhost:3000/sparql/
kegg	http://localhost:4000/sparql/
jamendo	http://localhost:5000/sparql/
swdogfood	http://localhost:5500/sparql/
dbpedia	http://localhost:7000/sparql/
drugbank	http://localhost:8000/sparql/
nytimes	http://localhost:9000/sparql/