TÜRKÇE DOKÜMANLAR İÇİN ÖZELLEŞTİRİLEBİLİR WEB TABANLI DİKEY ARAMA MOTORU

Aydın KILIÇ
Yüksek Lisans Tezi

Bilgisayar Mühendisliği Anabilim Dalı
Eylül, 2008

JÜRİ VE ENSTİTÜ ONAYI

Aydın KILIÇ'ın "**Türkçe dökümanlar için özelleştirilebilir web tabanlı dikey arama motoru**" başlıklı **Bilgisayar Mühendisliği** Anabilim Dalı, **Bilişim** Bilim Dalı'ndaki, Yüksek Lisans Tezi 13.08.2008 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı-Soyadı	**İmza**
Üye (Tez danışmanı):	**Yard.Doç.Dr.ÖZGÜR YILMAZEL**	**.....**
Üye	**: Prof.Dr. CAN AYDAY**	**.....**
Üye	**: Yard.Doç.Dr.CÜNEYT AKINLAR**	**.....**

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ÖZET

Yüksek Lisans Tezi

TÜRKÇE DOKÜMANLAR İÇİN ÖZELLEŞTİRİLEBİLİR WEB TABANLI DİKEY ARAMA MOTORU

Aydın KILIÇ

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Yard. Doç. Dr. Özgür YILMAZEL
2008, 81 sayfa

İnternet ortaya çıktığı günden buyana sürekli olarak genişlemiş, daha önce eşi benzeri görülmemiş büyüklükte bir bilgi denize dönüşmüş ve İnternet kullanımı gündelik yaşantımızın bir parçası haline gelmiştir. İnternet büyüdükçe bu devasa bilgi yığını içerisinden ihtiyaç duyulan bilgiyi arayıp bulmak da gittikçe önem kazanmıştır. İşte tam bu noktada arama motorları devreye girmiş ve bilgi erişimini kolaylaştırmışlardır. Bütün ağı indeksleyen ve yatay arama motorları olarak adlandırılan bu arama motorlarının genel bilgi ihtiyaçlarının karşılamada gösterdikleri başarıyı özel ilgi alanlarına yönelik bilgi ihtiyaçlarını karşılamada gösteremedikleri görülmüştür. Yatay arama motorları içeriğin büyük bölümünün yazıldığı İngilizce gibi birkaç dil dışında dokümanların yazıldığı dilin özelliklerini dikkate almamaktadırlar. Yatay arama motorlarının özel bilgi ihtiyaçlarını karşılamada karşılaştıkları güçlüklerin üstesinden gelmek amacıyla belirli bir alana yoğunlaşmış ve alanın özelliğine göre indeksleme, sorgulama ve sıralama algoritmaları kullanan dikey arama motorlarının kullanımı gündeme gelmiştir.

Bu tezde Türk dilinin özelliklerini dikkate alan, kullanıcılar tarafından özelleştirilebilir web tabanlı bir dikey arama motoru geliştirilmesi hedeflenmiştir. Sayfalardaki Türkçe karakterlerin doğru olarak işlenmesi, dokümanın yazıldığı dilin tanınması, sembolleştirilen metnin köklerinin bulunması sağlanarak Türkçe dokümanların daha etkin olarak indekslenmesi hedeflenmiştir. Geliştirilen dikey arama motorunu oluşturan alt bileşenler ve arama motorunun geneli ayrı ayrı test edilmiş ve arama motorunun bilgi erişim etkinliğini artırdığı tespit edilmiştir.

Geliştirilen dikey arama motoru genişletilebilir bir yapıya sahip olduğundan ihtiyaç halinde bütün alt bileşenleri geliştirilip özelleştirilebilmektedir.

Anahtar Kelimeler: Dikey Arama Motorları, Türkçe Bilgi Erişimi, Heritrix, Lucene, Dil Tanıma

ABSTRACT

Master of Science Thesis

WEB BASED CUSTOMIZABLE VERTICAL SEARCH ENGINE FOR TURKISH DOCUMENTS

Aydın KILIÇ

Anadolu University
Graduate School of Sciences
Computer Engineering Program

Supervisor: Asst. Prof. Dr. Özgür YILMAZEL
2008, 81 page

From the beginning Internet has been expanding constantly, transformed into an unprecedented sea of knowledge and Internet usage has become a part of daily life. As Internet is getting bigger, finding needed information among this huge source of information is becoming more important. At this point search engines has taken the burden and made information access easier. Despite their success on meeting general information needs, search engines - indexing whole web - which are called horizontal search engines, couldn't repeat the same success on meeting information needs concerning specific interest areas. Except few languages most web content is created, like English, horizontal search engines don't consider the characteristic of content language. To overcome the difficulties horizontal search engines facing meeting specific information needs use of vertical search engines focused to some specific area, using special indexing, searching and ranking algorithms suitable for specific area, came into question.

In this thesis, we aimed to develop a user customizable web based vertical search engine which considers characteristics of Turkish language. It is aimed to index Turkish documents more efficiently by processing the Turkish characters found in pages correctly, identifying the document language and stemming tokenized text. We tested both sub components of search engine and whole vertical search engine and saw increased information retrieval performance.

Because our vertical search engine has an extendable architecture all sub components can be extended and customized if needed.

Keywords: Vertical Search Engines, Turkish Information Retrieval, Heritrix, Lucene, Language Identification

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

SİMGELER ve KISALTMALAR DİZİNİ

Crawler : Toparlayıcı
Crawl : Toparlamak
URI : Universal Resource Identifier
Evrensel Kaynak Tanımlayıcısı
URL : Universal Resource Locator
Evrensel Kaynak Göstericisi
TREC : Text Retrieval Conference
Metin Erişim Konferansları
NLP : Natural Language Processing
Doğal Dil İşleme
HTML : Hyper Text Markup Language
Hiper Metin İşaretleme Dili
XML : Extensible Markup Language
Genişletilebilir İşaretleme Dili
PDF : Postscript Document File
IP : Internet Protocol
İnternet Protokolü
DNS : Domain Name Service
Alan Adı Sistemi
JMX : Java Management Extension
Java Yönetim Uzantısı
IA : Internet Archive

1. GİRİŞ VE AMAÇ

İnternet 90'lı yılların başında ortaya çıkmasının ardından olağanüstü bir hızla büyümeye başlamıştır. Bu büyüme halen devam etmektedir. İnternet daha önce hayal bile edilemeyen alanlara da yayılarak insanoğlunun sosyal yaşantısının bir parçası haline gelmeye başlamıştır. İnternetin barındırdığı bilginin çeşitliliği ve miktarı nedeniyle İnternetin ilk zamanlarında bile bu inanılmaz boyutlardaki bilgi yığını içerisinde elimizde bağlantısı olmayan kaynaklara erişmenin tek yolu arama motorları olmuştur.

Web birçok açıdan eşi benzeri görülmemiş bir yapıya sahiptir. Eşi benzeri görülmemiş bir ölçeğe, içeriğin yaratılmasında koordinasyon eksikliğine ya da koordinasyonsuzluğa, katkıda bulunanların altyapıları ve amaçları bakımından farklılığa sahiptir. Bütün bu faktörler, Web'in aranmasını klasik dokümanların aranmasından farklı kılmaktadır [1].

Yukarıda sayılan nedenlerden dolayı İnternette arama yapmanın da kendine has güçlükleri bulunmaktadır. Özellikle, içeriğinin ve bu içeriği sağlayan kaynağın güvenirliğini tespit etmenin son derece güç olduğu bir ortamda ihtiyaç duyulan bilgiyi içerisinde bulunduran sayfa ve dokümanlardan gerçekten hangilerinin ilgili olduklarını bulmak ve bunları mümkün olduğu kadar doğru şekilde ilgililik derecelerine göre sıralayarak kullanıcılara sunmak kolay bir iş değildir.

Yaygın olarak kullanılan arama motorları bütün ağı indekslemeye çalışmaktadırlar. İnternette karşılaşılan her tür bilgi için ayrı ayrı analiz ve işleme yöntemi geliştirilemeyeceğinden bu arama motorları bütün dokümanları genel bir yaklaşım kullanarak indekslemektedirler.

Kullandıkları genel yaklaşım nedeniyle, bu arama motorları yatay arama motorları olarak da adlandırılmaktadır. Yatay arama motorları genel bilgi ihtiyaçlarına cevap verse de, kullanıcıların belirli konulardaki ihtiyaçlarını karşılamada yetersiz kalmaktadırlar. Kullandıkları sıralama algoritmalar popüler kaynaklarda bulunan sonuçları üst sıralara taşıdığından popüler olmayan konularda yapılan aramalarda arzu edilen dokümanlar bulunmuş olsa bile çoğu zaman arama sonuçları içinde çok gerilerde kaldığında pratikte erişilememektedir.

Yatay arama motorlarının özel bilgi ihtiyaçlarını karşılamada karşılaştıkları güçlükler nedeniyle, belirli bir alana yoğunlaşmış ve alanın özelliğine göre indeksleme, sorgulama ve sıralama algoritmaları kullanan dikey arama motorlarının kullanımı gündeme gelmiştir. Hali hazırda çeşitli konularda özelleşmiş birçok dikey arama motoru mevcuttur.

Dikey arama motorları ile hedeflenen alanda daha fazla derinliğe ve daha yüksek ilgililik oranlarına ulaşmak amaçlanmaktadır. Dikey arama motorları aynı zamanda kullanıcılara düşük maliyetle ilgi alanlarında özelleşmiş arama motorlarına sahip olma imkanı da sağlayabilmektedir.

Bu çalışmada geliştirilen dikey arama motoru dokümanları indekslerken ve sorgularken Türk dilini özelliklerini dikkate almakta, kullanıcıların kendi ilgi alanlarındaki web sitelerini indekslemelerine, dokümanlar içerisinde bulunması istenen anahtar kelimeleri belirleyebilmelerine olanak sağlamaktadır. Ayrıca, arama motorumuz web tabanlı bir arayüz kullanılarak uzaktan yönetilebilmektedir.

Burada bir dikey arama motorunun geliştirilebilmesi için bilinmesi gereken bilgi erişimi, dikey arama ve doğal dil işleme konuların ne anlama geldiğini kısaca açıklanacaktır.

1.1. Bilgi Erişimi

Yazılı bilginin arşivlenmesi, Sümerlerin üzerinde çivi yazısı bulunan kil tabletlerini depolanmak için özel yerler tesis ettikleri milattan önce 3000'li yıllara kadar dayanmaktadır. O zamanlarda bile, Sümerler bilginin etkin olarak kullanılabilmesi için uygun şekilde organize edilmesi ve erişiminin sağlanması gerektiğinin farkına vardılar. Tabletleri ve içeriğini belirleyecek şekilde özel bir sınıflandırma sistemi geliştirdiler [2].

Özellikle kağıdın ve matbaanın icadında sonra yazılı bilginin depolanması ve bilgiye erişilmesi gittikçe daha önemli hale gelmiştir. Bilgisayarlar icat edildikten sonra insanlar bilgisayarların çok büyük miktarlardaki bilginin depolanmasında ve bilgiye erişimde kullanılabileceklerini fark ettiler.

Bilgi erişimi terimi çok geniş anlamda kullanılabilmektedir. Kredi kartının numarasını yazmak için cebinizden çıkarmak ve numarayı okumak bir tür bilgi

erişimidir. Ancak akademik anlamada bilgi erişimini şu şekilde tanımlayabiliriz. Bilgi Erişimi; bir bilgi ihtiyacına cevap veren, genellikle yapılandırılmamış bir tabiata sahip kaynakların (genellikle dokümanlar) büyük koleksiyonların arasından bulunup getirilmesidir [1].

Bilgi erişiminin ilk uygulamaları 1950 ve 1960'lı yıllarda gerçekleştirilmiştir. 1990'larda genellikle birkaç bin dokümandan oluşan koleksiyonlar üzerinde gayet iyi çalışan yöntemler geliştirilmiştir.

1992 yılında Amerikan Savunma Bakanlığı ve Ulusal teknoloji ve Standartlar Enstitüsünün desteğinde Text Retrieval Conference (TREC)'ları düzenlenmeye başlanmıştır [3]. Bu konferansların amacı, bilgi erişim camiasına geliştirdikleri bilgi erişim metotlarını çok büyük koleksiyonlar üzerinde test etmek ve değerlendirmek için bir altyapı sunmaktır. Bu konferanslar çok büyük koleksiyonlar üzerinde çalışabilecek metotların geliştirilmesinde katalizör görevi üstlenmiştir. Arama motorlarının ortaya çıkışı çok büyük çaplı erişim sistemlerine olan ihtiyacı artırmıştır.

Günümüzde arama motorları gibi bilgi erişim sistemleri günlük hayatımızın bir parçası haline gelmiştir. Bilgi erişim sistemleri İnternet çapında arama motorlarından, masaüstü arama yazılımlarına, çok çeşitli amaçlar için geliştirilmiş irili ufaklı birçok farklı bilgi erişim sistemini kapsar hale gelmiştir.

Bilgi erişim süreci kullanıcının sisteme bir sorgu girmesiyle başlar. Web arama motorlarında olduğu gibi sorgular bilgi ihtiyacını temsil eden ifadelerdir. Bilgi erişiminde bir sorgu genellikle koleksiyondan sadece bir nesne ile temsil edilmez. Genellikle sorguya farklı uygunluk derecelerinde birden fazla nesne karşılık gelir. Sorguya karşılık gelen nesnelerin gerçekte konu ile ne kadar alakalı oluğunun tespiti ve sonuçların olabildiğince mantıklı bir şekilde sıraya dizilmesi bilgi erişim sistemlerinin en temel problemlerinden birisini teşkil etmektedir.

1.2. Dikey Arama

Dikey Arama terimi aslında kolay anlaşılabilecek bir terim değildir. Dikey arama yerine "özelleştirilmiş arama" (specialized search) veya "uzmanlaşmış arama" (specialty search) terimleri de kullanılmaktadır. Dikey arama terimini daha çok finansal çevreler kullanmış ve popüler hale getirmiştir.

Google, Yahoo, Microsoft arama motorlarından herhangi birini kullanarak çok geniş bir spektrumdaki kaynak içerisinde yaptığınız aramalar "yatay" arama olarak adlandırılmaktadır. Sorgulama sonuçları spor siteleri, haber siteleri, sağlık siteleri, alışveriş siteleri gibi yatay spektrumun bütün konularını kapsamaktadır.

Dikey aramada ise, bu sorgu sonuçları belirli bir ilgi alanı ile sınırlandırılır. Sadece haber sitelerini veya sağlık sitelerini sorgularsınız. Bu tür bir odaklanma arama sonuçlarının ilgililik oranını artırmaktadır.

Evinizdeki pencereyi onarmaya ihtiyacınız varsa yatay arama motorlarında "fixing windows" ifadesini aradığınızda Internet'te Windows işletim sistemi hakkında çok fazla bilgi olduğundan Windows işletim sistemi hakkında bilgiler sonuç lisesinin büyük bir bölümünü oluşturacaktır. Eğer sorgulamayı sadece ev düzenleme ve tamiratı hakkında bilgiler içeren bir dikey arama motorunda yaparsanız bulduğunuz sonuçlar çok daha ilgili olacaktır [4].

Yatay arama her zaman kullanıcıların ihtiyaçlarına tam olarak cevap veremediğinden büyük arama motorları dahil bir çok firma çeşitli konularda dikey arama motorları geliştirmişlerdir. En büyük arama motorlarından birisi olan Google yaygın olarak bilinen ve kullanılan yatay arama motorunun yanında çeşitli konularda özelleşmiş bir dizi dikey arama motoru da işletmektedir. Bu dikey arama motorlarının listesi Çizelge 1.1'de görülebilir [4].

Çizelge 1.1 Google firmasının dikey arama motorları

Sıra	**Arama Motoru**	**Sıra**	**Arama Motoru**
1	Blog Search	8	Local/Maps
2	Book Search	9	News
3	Catalogs	10	Patent Search
4	Code Search	11	Product Search
5	Directory	12	Scholar
6	Finance	13	Video
7	Images	14	Web Search

Örneğin Google kitap arama motoru Google'ın tarayarak sayısallaştırdığı ve optik karakter tanıma yazılımları kullanarak metinlerini indekslediği kitapların

içerisinden arama yapılabilmesini sağlayan bir dikey arama motorudur. Bu arama motoruna istenilen anahtar kelimeler girilerek kitapların içeriğinde arama yapılabilmektedir. Sonuçların bulunduğu kitap sayfalarının ön izlemesi de görülebilmektedir.

Dikey arama motorları genel olarak üç farklı kategoriye ayrılabilirler [5].

- **Yerel**

  Belirli bir coğrafi bölge veya yerleşim birimi ile bilgilerin bulunduğu dikey arama motorlarıdır. Bu bilgiler bölgeye has gazeteleri, dergileri, tarihi ve turistik mekanları, sanat ve ticaret hayatı ile ilgili bilgileri kapsayabilir.

- **Konusal**

  İçeriği sağlık, eğitim, elektronik, otomobil ve spor gibi belirli bir konuda özelleşmiş dikey arama motorlarıdır.

- **Ticari**

  Ticari amaçlarla kurulmuş belirli bir sektöre yönelik daha çok profesyonel çalışanların konularıyla ilgili bilgileri aramada kullandığı dikey arama motorlarıdır. Üyelik gerektiren akademik veritabanları ve arama motorları bu kategoriye örnek olarak verilebilirler.

Arama sonuçları arama motorunun hizmet vermeyi amaçladığı belirli bir sektör veya alanla ilgili içerikten oluşturulur. Dikey arama motorlarının diğer arama motorlarından ayıran belli başlı özellikler şunlardır [5].

- Arama hizmeti sağlayıcısı belirli bir hedef içeriği kendisi belirler veya kullanıcılara seçme imkanı tanır.
- Arama hizmeti sağlayıcısı ilgili olmayan konuları ayıklamak için seçilen konu hakkındaki uzmanlığını kullanır.
- İçerik web gibi açık kaynaklardan, ticari kaynaklardan, destekçilerden, diğer kullanıcılardan veya bunların bir karışımından meydana gelebilir.
- Sağlanan arama çözümü ticari rehberler gibi yapısal veritabanlarına alternatif erişim metodu olabilir
- Sağlanan çözüm içeriğin karşılaştırılmasını sağlayarak katma değer yaratabilir.

Dikey arama motorları farklı kişilere farklı anlam etmektedirler. Bu nedenle dikey arama motorları belirli bir konuda yoğunlaşmış bir İnternet portalı, var olan bir web sitesine ilave bir web uygulaması veya daha çok teknik konularda kullanılan birçok parametreye göre arama yapabilen arama motorları şeklinde görülebilmektedir.

1.3. Doğal Dil İşleme

Doğal Dil İşleme, yaygın olarak bilinen adıyla NLP (Natural Language Processing), yapay zeka ve dil biliminin bir alt kategorisidir. Türkçe, İngilizce, Almanca, Fransızca gibi doğal dillerin işlenmesi ve kullanılması amacı ile araştırma yapan bilim dalıdır.

Doğal Dil İşleme, doğal dillerin kurallı yapısının çözümlenerek anlaşılması veya yeniden üretilmesi amacını taşır. Bu çözümlemenin insanlara sağlayacağı kolaylıklar, yazılı dokümanların otomatik çevrilmesi, soru-cevap makineleri, otomatik konuşma ve komut anlama, konuşma sentezi, konuşma üretme, otomatik metin özetleme, bilgi sağlama gibi birçok başlıkla özetlenebilir. Bilgisayar teknolojisinin yaygın kullanımı, bu başlıklardan bazılarını tamamen veya kısmen gerçekleştirebilen yazılımların günlük hayatımızın bir parçası haline gelmesini sağlamıştır. Örneğin, hemen hemen bütün kelime işlemciler birer imla düzeltme aracı ile beraber gelmektedir. Bu araçlar aslında yazılan metni çözümleyerek dil kurallarını denetleyen doğal dil işleme yazılımlarıdır.

Doğal dilin anlaşılması dış dünya hakkında son derece geniş kapsamlı bir bilgiye ve bu bilginin doğru şekilde kullanılması ve yorumlanmasına ihtiyaç duyduğundan tam bir yapay zeka problemi olarak görülebilir.

Doğal dil işlemede karşılaşılan problemlerden bazılarına burada çok kısaca değineceğiz [6]

- **Konuşmanın parçalara ayrılması**

Birçok konuşma dilinde birbirini takip eden harfleri temsil eden sesler birbirlerine karışırlar. Bu nedenle analog ses sinyalinin karakterlere dönüştürülmesi son derece zor olabilir. Ayrıca günlük konuşma dilinde konuşmalar arasında hemen hemen hiç boşluk bırakılmaz. Konuşmanın gramatik ve anlamsal olarak çözülebilmesi için kelime ve cümlelerin başlangıç ve

bitimlerinin tespit edilmesi ve kullanıldıkları bağlamın da bilinmesi gerekmektedir.

- **Metnin parçalara ayrılması**

Konuşma ve yazı dilinde bir cümlenin başladığı ve bittiği yeri tespit etmek son derece güç olabilir. Özellikle metnin anlamının doğru şekilde çözülmesi için öncelikle analiz için hedef teşkil edecek olan kelime öbeklerini yani cümlenin doğru şekilde tespit edilmesi gereklidir. Buna ilave olarak Çince ve Japonca gibi bazı yazı dillerinde kelime sınırları da bulunmamaktadır. Bu nedenle kelimelerin sınırlarının belirlenmesi kolay bir işlem değildir.

- **Kelimelerin birden fazla anlam ifade etmesi**

Birçok kelimenin birden fazla anlamı vardır. Kelimenin geçerli bağlamada hangi anlamda kullanıldığının tespit edilmesi gerekmektedir. Türkçe için bir örnek vermek gerekirse aşağıdaki cümlelerde "yüz" kelimesi farklı anlamlarda kullanılmıştır.

Denize gidince bol bol **yüz** tamam mı?

Cilt bakımında **yüz** temizliğine önem verilmelidir.

Sence **yüz** lira fazla değil mi?

- **Sözdizimsel anlam farklılıkları**

Doğal dillerin sözdizimleri de cümle veya kelimelerin birden fazla anlama gelmesine neden olabilmektedir. Konuşma dilinde cümle sınırlarının farklı belirlenmesi anlamın farklılaşmasına neden olabilmektedir.

- **Mimik ve Jestler**

Konuşma dilinde mimik ve jestler en az konuşmacının ifade ettiği kelimeler kadar önemlidir. Konuşulan kelimelerin anlamlarını kısmen veya tamamen değiştirebilirler.

Bu tez çalışmasında önemli olan konu kelimelerin köklerinin mümkün olduğunca doğru olarak bulunması ve bilgi erişiminin kalitesinin artırılması olduğundan, ileriki bölümlerde kelime köklerinin bulunması konusuna daha ayrıntılı şekilde değinilecektir.

2. ALT YAPI VE İLGİLİ ÇALIŞMALAR

Bir arama motoru birçok bileşenden oluşur ancak, arama motorlarını temel olarak üç alt bileşene ayırmak mümkündür. Bu bileşenler şunlardır;

- Toparlayıcı
- İndeksleme bileşeni
- Sorgulama bileşeni

Bu bölümde arama motorumuzun alt bileşenleri ve bu çalışmada kullanılan arama motorunun Türkçe dokümanları etkin şekilde tanıması, indekslemesi ve sorgulaması için gerekli altyapı hakkında daha ayrıntılı bilgiler verilecektir.

2.1. Toparlayıcı

En temel anlamıyla toparlayıcı genellikle küçük bir grup Evrensel Kaynak Tanımlayıcısı (Universal Resource Identifier) yaygın kullanılan kısaltmasıyla URI'den başlayarak bu URI'lerin işaret ettiği dokümanları indiren, bu dokümandan diğer URI'lere olan bağlantıları çıkaran, tespit ettiği bağlantıların işaret ettiği dokümanları tekrar indiren ve bunlardan diğer dokümanlara işaret eden URI'leri tekrar çıkaran ve bu işleme sürekli olarak devam eden bir yazılımdır.

Geliştirdiğimiz dikey arama motorunda toparlayıcı olarak İnternetin arşivlenmesini amaçlayan ve kar amacı gütmeyen bir kurum olan İnternet Archive'ın geliştirdiği Heritrix olarak adlandırılan toparlayıcı kullanılmıştır.

Pratikte toparlama işlemi genellikle yukarıda kısaca bahsettiğimiz kadar basit değildir. Öncelikle bağlantıların bulunması son derece zor olabilir. Genellikle HTML sayfaların işlenmesi kolaydır. Sayfadaki *a* etiketinin *href*, *img* gibi birkaç etiketin *src* özniteliğinin değerlerini bulmak yeterlidir. Ancak, gerçekte bağlantılar her zaman kolaylıkla okunan HTML etiketlerinin öznitelikleri olarak kodlanmamıştır. JavaScript gibi betik dilleri bağlantıları çalışma zamanında otomatik olarak oluşturabilmektedir. Ayrıca bağlantılar Word, Excel, Powerpoint, Adobe Flash ve PDF gibi dosya tipleri içine dosya formatına göre değişiklik gösteren bir biçimde gömülmüş olabilmektedir.

Diğer bir problem nezaket kurallarına uyulması gerekliliğidir. Toparlayıcının hedef web sunucudan ardı ardına sürekli olarak talepte bulunması muhtemelen sunucunun cevap veremez hale gelmesine neden olacaktır. Dolayısıyla yapılan taleplerin sayısına yönelik olarak bir çeşit kontrol mekanizması geliştirilmesi zorunludur. Birbirini takip eden iki talep arasına belirli bir zaman aralığı konması bunu gerçekleştirmenin yöntemlerinden birisidir. Ancak bu zaman aralığı her sunucu için ayrı ayrı olacak şekilde ayarlanmazsa toparlama hızında ciddi bir düşüşe neden olacaktır. Bu nedenle, her bir web sunucu için ayrı bir URI kuyruğu tutulması gereklidir.

Bazı alan adları birden fazla web sunucusu ile veya birden fazla alan adı bir web sunucu ile hizmet verebileceğinden bu işlem biraz karmaşık olabilir. Genel olarak ya alan adı ya da IP adresi bazında bir limit belirlenerek nezaket kuralına uyulması sağlanır. Her iki yaklaşım da mükemmel değildir ancak genellikle web sunucuların üzerine bindirilen yükü makul seviyelerde tutabilirler.

Toparlayıcı geliştirilirken Robot Exclusion standartlarına uyulması, karakter kodlamalarının doğru olarak tespit edilmesi, takvimler gibi dinamik olarak üretilen ve sürekli uzayıp giden bağlantılarla başa çıkılması gibi birçok hususun da göz önüne alınması gereklidir.

İnternet sürekli olarak büyümekte ve hızlı bir biçimde değişmektedir. Bu değişim içeriğin kısa sürede güncelliğini kaybetmesine neden olmaktadır. Bir dokümandan bağlantının çıkarıldığı zaman ile bağlantının işaret ettiği kaynağın ziyaret edilmesi arasında geçen süre uzun olursa bağlantının işaret ettiği kaynak değişmiş, kaldırılmış veya tamamen farklı bir kaynak olabilir. Bu nedenle, tespit edilen bir kaynak uygun bir süre içerisinde ziyaret edilmelidir. Ancak geniş çaplı toparlamalarda bu mümkün olmayabilir.

Eğer toparlayıcıya bir limit belirlenmezse, tespit edilen kaynaklar kullanılan donanımın kaynakları tükenene kadar sürekli olarak artacaktır. Bu nedenle, toparlamanın kapsamının ya bütün siteler için bağlantı derinliğinin ya web sitelerinin sayısının ya da her ikisinin birden sınırlanması gibi bir yöntemle daraltılması gerekecektir.

Her bir web sitesinden indirilecek doküman sayısını sınırlamak genel olarak kullanılan bir yöntemdir. Ancak bu çalışmada amaç seçilen web

sitelerinden belirlenen kıstaslara uyan azami dokümanı indekslemek olduğunda bu tür bir kısıtlamaya gidilmeyecektir. Olabildiğince fazla doküman bulunmaya ve indirip indekslemeye çalışacaktır.

Toparlamanın kapsamı ve seçilecek yöntem toparlamanın amacına bağlı olarak değişiklik gösterir. Heritrix'i geliştiren takım Heritrix'in kullanılabileceği ya da başarılı olabileceği dört tür toparlamayı vurgulamaktadırlar [7].

- **Geniş kapsamlı toparlama:** Web sitelerden toparlanan sayfaların ve kaynakların bütünlüğü kadar web sitelerinin sayısının da önemli olduğu yüksek bant genişliği gerektiren toparlamalardır. En aşırı uçta geniş kapsamlı toparlama zaman, kaynaklar ve bant genişliğinin yeterli olduğu durumlarda İnternetin olabildiğince büyük bir kısmını toparlamaya çalışır.
- **Odaklanmış toparlama:** Kalite kriterinin seçilmiş bazı web siteleri ve konuların tamamının kapsanması olduğu, küçük ve orta ölçekli (genellikle 10 milyon eşsiz dokümandan az) odaklanmış toparlamalardır.
- **Sürekli toparlama:** Geleneksel olarak toparlamalar ilgilenilen konulardaki kaynakların bir defaya mahsus o anki durumlarını kaydetmeyi hedeflerler. Sürekli toparlama bu durumun aksine daha önce indirilen sayfaları sürekli olarak tekrar ziyaret ederek değişiklikleri veya yeni eklenen sayfaları tespit etmeyi ve indirmeyi amaçlayan ve ziyaret sıklığını operatörün belirlediği veya hesaplanan güncelleme sıklığına göre ayarlayabilen bir toparlama türüdür.
- **Deneysel toparlama:** Farklı protokoller ile çeşitli teknikler kullanılarak yapılabilecek bir toparlama türüdür.

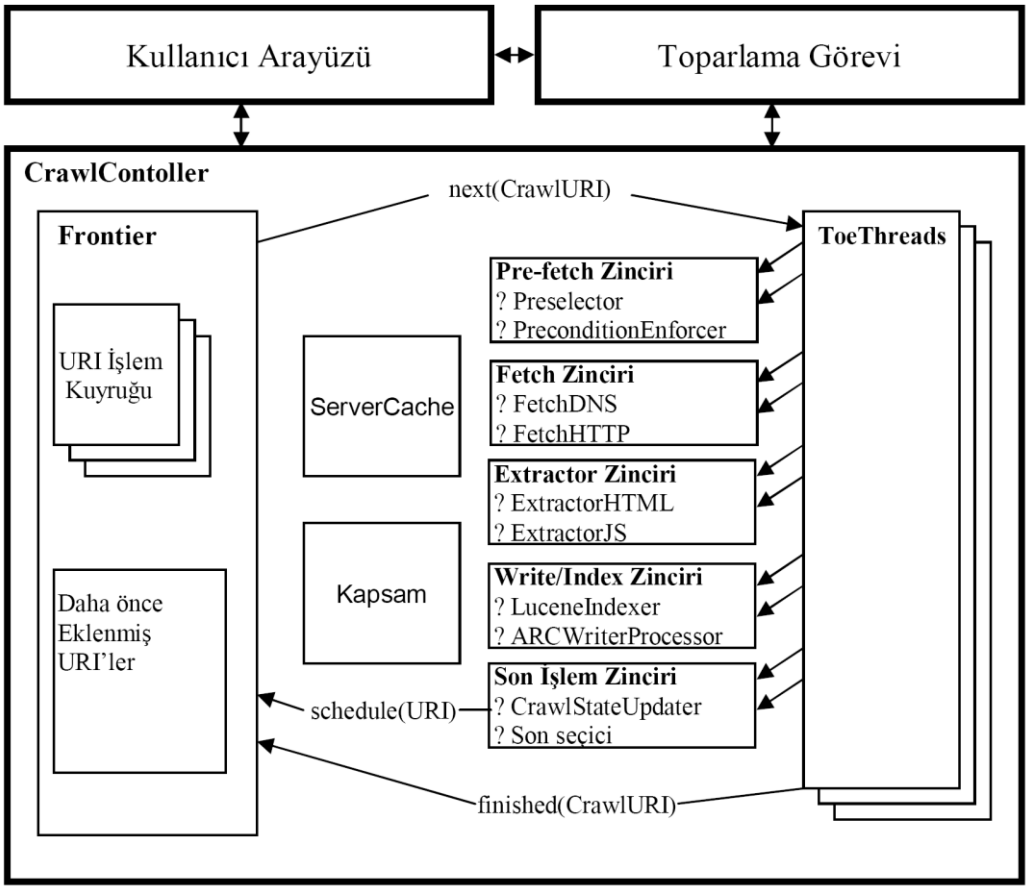
2.1.1. Toparlayıcıya genel bir bakış

İleriki bölümlerde izah edilen konuların daha kolay bir şekilde anlaşılabilmesi için öncelikle Heritrix toparlayıcısının genel olarak nasıl çalıştığının anlaşılmasında fayda vardır. Heritrix'in mimarisi ve işleyişinin

hakkındaki bilgiler Heritrix kullanıcı ve geliştirici dokümanlarından alınan bilgilerden faydalanılarak hazırlanmıştır [8].

Heritrix toparlayıcısı modüler bir yapıda tasarlanmıştır. Çalışma zamanında hangi modüllerin kullanılacağı belirlenebilmektedir. Toparlayıcının varsayılan davranışından farklı bir şekilde kullanılabilmesi için standart yapılandırmada gelen modüllere ilave olarak veya bu modüllerin yerine ihtiyaca göre yazılmış yeni modüllerin kullanılması yeterli olmaktadır.

Toparlayıcı temel sınıflar ve eklenebilir modüllerden oluşmaktadır. Temel sınıflar yapılandırılabilirler ancak değiştirilemezler. Eklenebilir modüller ise toparlayıcının yapılandırması değiştirilerek eklenip çıkarılabilirler. Birkaç temel eklenebilir modül toparlayıcı ile beraber gelmektedir, ancak bu modüller ihtiyacı karşılamıyorsa istenilen modül yazılarak toparlayıcıya eklenebilir. Toparlayıcının genel görünümü Şekil 2.1'de görüldüğü gibidir. Burada kısaca Şekil 2.1'de görülen toparlayıcının bileşenleri açıklanacaktır.



Şekil 2.1 Toparlayıcının genel görünümü

- **CrawlController**

CrawlController toparlama işleminin yapılması için birlikte çalışan bütün sınıfları bünyesinde barındırır, çalışan toparlamaya üst seviye bir arabirim sağlar ve Frontier'dan ToeThread'lere URI'leri dağıtan ana kanalı çalıştırır. Toparlama için global bağlam olarak alt bileşenler genellikle birbirlerine CrawlController vasıtasıyla ulaşırlar.

- **Frontier**

Frontier toparlanacak bir sonraki URI'nin dağıtılmasından ve web sunucuya fazla yük bindirilmemesi için nezaket kuralının uygulanmasında sorumludur. Bir URI toparlandıktan sonra aynı URI keşfedilen yeni URI'ler ile birlikte toparlama için planlanmak üzere tekrar Frontier'a devredilir.
Toparlamanın durumunu Frontier takip eder. Aşağıdaki liste takip edilen hususlardan bir kaçını göstermektedir.

  - Hangi URI'ler tespit edilmiştir.
  - Hangi URI'ler işlenmektedir.
  - Hangi URI'ler işlenmiştir.

Frontier, Frontier arayüzünü gerçekleştirmektedir ve bu arayüzü gerçekleştiren herhangi bir Frontier ile değiştirilebilmektedir. Bir Frontier yazmanın pek kolay bir iş olmadığını hatırlatmakta yarar var.

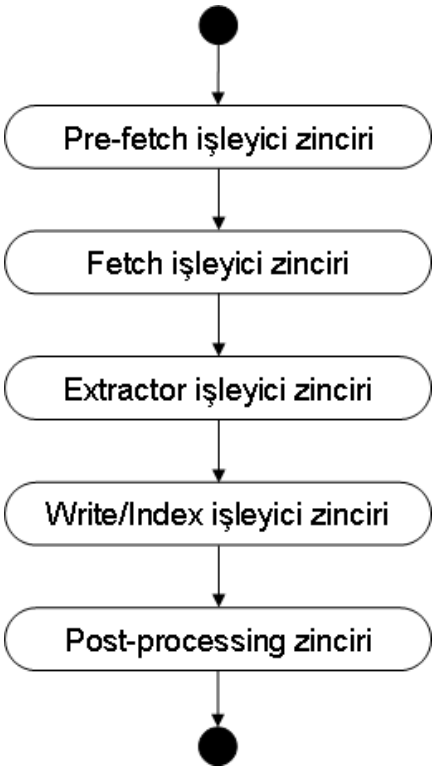
Frontier en azından PreconditionEnforcer, LinksScoper ve FrontierScheduler harici süreçlerinin davranışlarına dayanmaktadır. PreconditionEnforcer indirme işleminden önce DNS'in ve robots. txt dosyalarının kontrol edildiğinden emin olunmasını sağlar. LinksScoper ilgilendiğimiz belirli bir URI'nin toparlama kapsamında olup olmadığına, o URI'nin bizim için ne kadar önemli olduğuna ve hangi öncelik derecesiyle indirileceğine karar verir. FrontierScheduler URI'leri toparlamak üzere Frontier'a ekler.

- **ToeThread'ler**

Heritrix toparlayıcısı çok kanallı bir uygulamadır. Her bir URI ToeThread olarak adlandırılan kendi kanalında ele alınır. ThoeThread Frontier den yeni bir URI talep eder, bunu sırasıyla bütün işleyiciden geçirir ve tekrar yeni bir URI talep eder.

- **İşleyiciler**

İşleyiciler Şekil 2.2'de görüldüğü gibi işleyici zincirleri şeklinde gruplanmışlardır. Her bir zincir URI üzerinde tanımlanan işlemleri gerçekleştirir. Bir işleyicinin URI ile ilgili işi tamamlandığında ThoeThread bütün işleyiciler tamamlanıncaya kadar URI'yi bir sonraki işleyiciye gönderir. Bir işleyicinin URI'ye belirli bir işleyici zincirini atlamasını söyleme seçeneği vardır. Ayrıca bir işleyicide onulmaz bir hata oluşursa işlem Post-Process zincirine gönderilir.



Şekil 2.2 İşleyici zincirleri

Şekil 2.2'de görülen işleyici zincirleri tarafından yerine getirilen görevler aşağıdadır:

- **Pre-fetch işleyici zinciri**

İlk zincir URI'nin bu noktada toparlanıp toparlanmayacağının incelenmesinden sorumludur. Bu inceleme DNS çözümlemesi, robots.txt dosyasının okunması ve kimlik doğrulaması gibi bütün ön koşuların karşılanıp karşılanmadığının kontrol edilmesini kapsar. Kapsam kontrolünden geçemeyen bir URI'nin toparlanmasının tamamen engellenmesi de mümkündür.

Pre-fetch işlem zincirinde aşağıdaki veya benzer işlemleri yerine getiren işleyiciler bulunmalıdır.

- **Preselector**

URI'nin gerçekten toparlanıp toparlanmayacağının denetlendiği son kontroldür. Örneğin kapsamı yeniden kontrol eder. Eğer toparlama başlangıcından sonra kapsam ile ilgili kurallar değiştirildiyse yapılan değişikliklerin uygulanmasını sağlar. Kapsam genellikle yeni URI'ler toparlanmak üzere Frontier'a eklenmeden önce LinksScoper tarafından kontrol edilir. Eğer kullanıcı kapsam limitlerini değiştirirse bu değişiklik hali hazırda kuyrukta bekleyen URI'leri etkilemez. Bu noktada yapılan kapsam kontrolü sadece geçerli kapsam dahilindeki URI'lerin toparlanmasını sağlar.

- **PreconditionEnforcer**

Bir URI'nin toparlanabilmesi için gerekli olan DNS çözümlemesi ve robots.txt bilgilerinin okunması gibi ön koşulların yerine getirilip getirilmediğini kontrol eder.

- **Fetch işleyici zinciri**

Bu zincirdeki işleyiciler verinin uzaktaki sunucudan alınmasından sorumludurlar. Heritrix'in desteklediği her bir protokol için bir işleyici bulunmalıdır. Örneğin `FetchHTTP` Heritrix'in HTTP protokolünü destekleyen işleyicisidir.

- **Extractor işleyici zinciri**

Bu zincirde URI'nin işaret ettiği dokümanın içeriği sırasıyla işleyicilerden geçirilerek dokümanda bulunun bütün bağlantılar çıkarılmaya çalışılır.

- **Write/Index işleyici zinciri**

Bu zincir verinin arşiv dosyalarına yazılmasından sorumludur. Heritrix veriyi ARC formatında yazan `ARCWriterProcessor` işleyicisi ile beraber gelmektedir [9]. Verinin başka formatlarda yazılması, veritabanlarına kaydedilmesi, indekslenmesi ve farklı amaçlarla kullanılabilmesi için yeni işleyiciler yazılıp Heritrix'e eklenebilir.

- **Post-processing işleyici zinciri**

Bütün URI'ler daha önce bir işleyici tarafından toparlanmamasına karar verilmiş olsalar dahi bu zincirden geçmek zorundadır. Post-processing zinciri aşağıdaki veya benzer görevleri yerine getiren modülleri bulundurmak zorundadır.

- **CrawlStateUpdater**

URI'nin işaret ettiği dokümanın indirilmesi esnasında tespit edilebilecek bilgilerle alan adı bazında IP adresi ve robots.txt bilgilerini günceller.

- **LinksScoper**

İndirilen dokümandan çıkarılan bağlantılarla toparlama kapsamını karşılaştırır. Kapsam dışında kalanlar dikkate alınmazlar. Ancak dikkate alınmayan URI'lerin kaydedilmesi sağlanabilir.

- **FrontierScheduler**

İndirilen dokümanlardan tespit edilen toparlamaya aday URI'leri toparlama için planlar. Eğer varsa URI için ön koşulların planlanmasını da yapar.

Heritrix'in genel olarak açıklanan mimarisinden bahsettikten sonra yukarıda kısaca değinilen konular daha detaylı bir şekilde açıklanacaktır.

2.1.2. Toparlayıcının anahtar bileşenleri

- **CrawlController**

CrawlController Hertirix altyapısının çekirdeğinde bulunmaktadır. Esasında toparlama işlemini o yürütür. Bir toparlama oluşturulduğunda ilk olarak bir CrawlController nesnesi yaratılır. CrawlController sırasıyla kendisine verilen yapılandırmaları okur ve toparlama için gerekli bütün modülleri oluşturur. Ayrıca CrawlController ToeThread'leri yaratıp Frontier ile çalışacak şekilde ayarlayarak onları yönetir.

Heritrix'in oluşturduğu bütün temel günlük kayıtları burada yaratılır ve diğer bileşenler bu kayıtlara CrawlController vasıtasıyla erişirler. Aslında neredeyse bütün bileşenlerin doğrudan veya ileride açıklanacak yapılandırma altyapısı vasıtasıyla dolaylı olarak CrawlController'a erişimleri vardır.

CrawlController gerekli hazırlıkları yaptıktan sonra toparlamanın başlatılması için komut bekler. CrawlController başlatıldıktan sonra duraklatılabilir, devam ettirilebilir veya toparlama sona erdirilebilir. Bu işlemler genellikle kullanıcı arabirimi tarafından gerçekleştirilir.

CrawlController genel bir arayüz kullandığından Heritrix için bir modül yazılırken yeni modül mevcut modüllere benzer bir şekilde CrawlController ile

etkileşimde bulunacağından CrawlController ile etkileşim kolaylıkla sağlanabilir. Önemli olan modüllerin CrawlController'ın toparlamayı kontrol etmekte kullandığı yordamları doğru şekilde gerçekleştirmeleridir.

- **ToeThread'ler**

Heritrix çok kanallı bir mimariye sahiptir ve keşfedilen URI'leri işlemekte bir işçi kanal havuzu (pool of worker threads) kullanmaktadır. Bu işçi kanallar genellikle ToeThread'ler olarak adlandırılır. ToeThread'ler Frontier'dan URI'ler talep ederler. Eğer varsa Frontier sıradaki URI'yi verir ve ToeThread bu URI'yi bir dizi işleyici zincirinden geçirir. Bütün işleyicilerden geçen URI tekrar Frontier'a iade edilir ve ToeThread işlemek için yeni bir URI talebinde bulunur.

ToeThread'lere duraklama, devam etme ve iptal etme sinyalini göndermek Frontier'in görevidir. Bu işlem genellikle bir ToeThread yeni bir URI talebinde bulunduğu zaman yapılır. Eğer duraklama sinyali gönderildiyse Frontier ToeThread'leri bekletir. Gönderilen sinyal toparlamanın iptal edilmesi ise Frontier EndedException istisnası oluşturur ve ToeThread'leri sonlandırır.

- **Frontier**

Frontier toparlayıcının en önemli parçasını teşkil etmektedir. Her ne kadar CrawlController toparlayıcının günlük kayıtlarına erişim ve ToeThread'lerin yaratılması gibi görevlerini yerine getirse de toparlamayı asıl olarak Frontier yönetmektedir.

Her Frontier hangi URI'ler ile karşılaşıldığını bilmektedir. Frontier hangi URI'lerin hangi sırada toparlanmak için beklediğini belirleyen bir liste tutmaktadır. ToeThread'ler Frontier'dan bir URI talep eder, onu işlerler ve tespit ettikleri bağlantıları toparlanmak üzere planlar ve işlenen URI'yi tekrar Frontier'a geri verirler. Bu durum toparlamanın işleyişinin tamamen Frontier tarafından kontrol edildiğini göstermektedir.

Heritrix altyapısı sadece belirli bir Frontier sağlamakla kalmamakta, Frontier'lar için bir arayüz tanımlayarak yeni Frontier'ların geliştirilmesine ve mevcut olanın da değiştirilmesine imkan sağlamaktadır. Yazılacak bir Frontier bu arayüzü gerçekleştirmeli ve ayrıca Heritrix'in yapılandırma altyapısına erişebilmek ModuleType sınıfından türetilmelidirler.

Frontier'ın nasıl çalıştığını anlamak için Frontier arayüzünün en önemli yordamlarını kısaca incelemekte yarar vardır. İki temel yordamdan birisi toparlanmak üzere sıradaki URI'yi döndüren `next()` diğeri bir URI'yi alıp toparlanmak için planlayan `schedule(URI)` yordamlarıdır. İlave olarak Frontier'ın atadığı URI'nin işlenmesinin tamamlandığını bildirmekte `finished(CrawlURI)` yordamı kullanılmaktadır.

Eklenen bütün URI'lerin sıralanmasından Frontier sorumludur. Her Frontier kendi toparlama amacına uygun olarak URI'leri istediği herhangi bir ölçüte göre sıralayabilir.

Frontier ayrıca Frontier arayüzü tarafından tanımlanmış yordamlarla erişilen çalışma zamanıyla ilgili birkaç istatistiki bilgi de sağlamaktadır.

- Keşfedilen URI ler

Tespit edilen ve toparlama için planlanan eşsiz URI'lerin toplam sayısı.

- Sırada bekleyen URI'ler

ThoThread'lere atanmak için sırada bekleyen URI'lerin sayısı.

- Tamamlanmış URI'ler

ThoThread'lere atanmış ve tamamlanmış URI'lerin toplam sayısı. Tespit edilebilen hatalar döndüren URI'ler yeniden planlamaya alındığı için bu sayıya dahil değildir.

- Başarıyla işlenen URI'ler

Başarıyla işlenen URI'lerin toplam sayısı. HTTP hata kodları döndüren URI'ler de başarılı olarak kabul edilmektedir.

- İşlenmesi başarısız olan URI'ler

Herhangi bir sebepten dolayı işlenmesi başarısız olan URI'lerin toplam sayısı.

- Dikkate alınmayan URI'ler

Robots.txt kuralları gibi herhangi sebepten dolayı dikkate alınmayan URI'lerin toplam sayısı

- Toplam yazılan bayt

İndirilen bütün dokümanlarının bayt olarak toplam büyüklüğü.

- **Filtreler**

Filtrelerin asıl fonksiyonu bir URI'yi alıp belirli bir filtreye uyup uymadığını göre doğru/yanlış kararının verilmesidir. Filtreler kapsamlara, işleyicilere ve özel amaçlar için geliştirilen bütün modüllere uygulanabilmektedir. Örneğin `HttpFetcher` işleyicisi HTTP başlıkları okunduktan sonra indirme işlemine devam edilip edilmeyeceğine karar vermek için filtreler kullanmaktadır. Filtrenin etkisi yerleştirildiği yere ve sıraya göre farklılık gösterecektir. Bazen URI'nin kabul edilmesine bazen de reddedilmesine neden olacaktır.

Heritrix birçok genel amaçlı filtrenin yanı sıra özellikle kapsamın belirlenmesinde kullanılan özelleştirilmiş filtreler de sağlamaktadır. Genel filtrelere örnek olarak bir URI metninin ve içerik türünün düzenli ifadeler kullanılarak kontrol edildiği fitreyi verebiliriz. Daha özel filtreler arasında URL'de bulunan taksim sayısına bakarak yol derinliği kontrol eden filtreleri sayabiliriz.

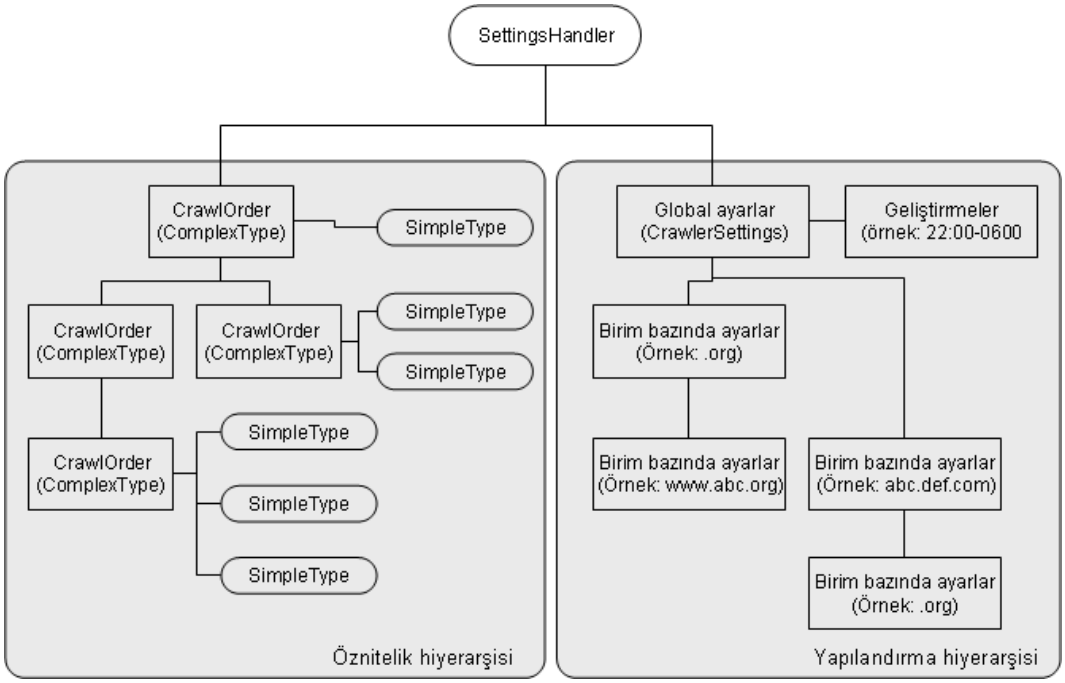
2.1.3. Yapılandırma altyapısı

Heritrix geniş kapsamlı ve esnek bir yapılandırma altyapısına sahiptir. Bu altyapı herhangi bir modülün çeşitli ayarlanabilir parametreler belirlemesine imkan tanımaktadır. Ayrıca bu parametrelerin değerleri alan adı ve birçok farklı kritere göre değişiklik göstertebilmektedir. Özünde yapılandırma altyapısı toparlayıcının herhangi bir sınıfı için bağlam duyarlı yapılandırma bilgilerinin kalıcı olarak saklanmasını sağlayan bir sistemdir.

Toparlayıcının yapılandırılabilir ayarları bulunan bütün sınıfları `ComplexType` veya bunun alt sınıflarından birinden türetilirler. `ComplexType` `javax.management.DynamicMBean` arayüzünü gerçekleştirmektedir. Bu nesnelere hangi öznitelikleri desteklediklerini sorgulayabilme ve bu öznitelikleri okumak ve değiştirebilmek için standart yöntemler kullanabilme olanağı sağlamaktadır.

Yapılandırma altyapısının giriş noktasını `SettingsHandler` soyut sınıfı oluşturmaktadır. Ayarların kalıcı depolama birimine yazılması ve okunmasından ve yapılandırma altyapısının farklı parçalarının birbirlerine bağlamasından bu

sınıf sorumludur. Toparlayıcının yapılandırma altyapısının şematik görünümü Şekil 2.3'de görüldüğü gibidir.



Şekil 2.3 Yapılandırma altyapısının şematik görünümü

2.1.4. Yapılandırma hiyerarşisi

Yapılandırma altyapısı hiyerarşik bir yapıya sahiptir. Bu hiyerarşi bir yapılandırma dosyasını temsil eden `CrawlerSettings` nesnesi ile oluşturulmaktadır. En tepede global ayarları temsil eden bir yapılandırma nesnesi bulunmaktadır. Bu nesne toparlama görevinin yürütülebilmesi için gerekli bütün ayarları barındırmaktadır. Global nesnenin altında her bir sunucu/alan adı için global ayarların yerini alacak sunucu/alan adına özgü ayarları barındıran birim bazında yapılandırma nesnesi bulunmaktadır.

Yapılandırma altyapısından belirli bir sunucu için bir özniteliğin değeri istendiğinde, ilk önce o sunucu için bu özniteliğin tanımlanıp tanımlanmadığına bakılmaktadır. Eğer tanımlanmış ise, özniteliğin değeri döndürülmektedir. Eğer tanımlanmamış ise, özniteliği bulana kadar sırasıyla birer seviye yukarı çıkarak aramaya devam edecek, en nihayetinde görev nesnesine kadar gidecektir. Eğer burada da bulunamaz ise, varsayılan değer döndürülecektir.

Bütün sunucu/alan adı yapılandırma nesneleri sadece o sunucu/alan adı için farklı olan değerleri bulundurur. Geleneksel olarak en tepedeki nesne global yapılandırma ve bunun altındaki nesnelerde "birim bazında yapılandırma" veya "bastırmalar" olarak adlandırılırlar.

Resim biraz daha karmaşık hale getirilirse yukarıdakilere ilave olarak geliştirmeler olarak adlandırılan yapılandırma nesneleri de bulunmaktadır. Global veya birim bazında yapılandırma nesnelerine ait bu türden bir nesne, belirli bir kriter karşılandığında sahibinin yapılandırması yerine geçer. Bu kriterler işlenen URI'nin belirli bir düzenli ifadeye uyması veya günün belli bir zamanında ve belirli bir süre ile geçerli olan ayarlar şeklinde olabilirler.

2.1.5. ComplexType hiyerarşisi

Toparlayıcının yapılandırılabilir bütün modülleri `ComplexType` veya onun türetilmiş sınıflarında birini genişletmek zorundadırlar. Yapılandırılabilir bir modülün parametrelerin tanımlarının tutulmasından `ComplexType` sorumludur. Gerçek değerler ise, Java `HasMap` sınıfından türetilen bir `DataContainer` nesnesinde tutulmaktadır. Kullanıcılar doğrudan `DataContainer`'a ulaşmazlar, bunu yerine öznitelikleri okumak için `ComplexType` sınıfının yordamlarını kullanırlar.

Okumak veya yazmak için bir özniteliğe URI vasıtasıyla erişilirken doğru bağlamdaki özniteliğin okuduğundan veya değiştirdiğinden emin olunmalıdır. Bunu başarabilmek için yapılandırma nesnesini parametre olarak alan `getLocalAttribute(CrawlerSettings settings, String name)` ve `setAttribute(CrawlerSettings settings, Attribute attribute)` yordamları bulunmaktadır. Bu yordamlar sadece parametre olarak gönderilen yapılandırma nesneleri üzerinde işlem yaparlar. Java JMX standardına uyumluluk için ilave olarak `getAttribute(name)` ve `setAttribute(Attribute attribute)` yordamları da bulunmaktadır. Bu yordamlar her zaman global yapılandırma nesnesi üzerinde işlem yaparlar.

Bir toparlama görevi içerisinden istenilen bir özniteliğin okunması, öznitelik geçerli bağlamda mevcut olmasa bile özniteliğin değerini bulup okumak istediğinizden dolayı biraz farklılık gösterir. Yapılandırma altyapısı özniteliğin

değerini bulmak için yapılandırma hiyerarşisinde adım adım yukarı çıkarak bir değer bulmaya çalışmaktadır. Doğru bağlamın kullanıldığından emin olmak için `getAttribute(String name, CrawlURI uri)` metodu kullanılmalıdır.

Her bir özniteliğin bir tipi vardır. Kullanılabilecek tipler `Type` sınıfının altsınıflarından birisi olabilir. Heritrix yapılandırma altyapısında üç temel tip bulunmaktadır. Bular;

1. `SimpleType`
2. `ListType`
3. `ComplexType`

- **SimpleType**

`SimpleType` genellikle Java dilinin temel veri tipleri için bir paketi (wrapper) temsil etmektedir. Temel tiplere ilave olarak `java.util.Date` tipi ve Heritrix `TextField` tipini de desteklemektedir.

- **ListType**

`ListType` Java temel tiplerinden oluşan `DoubleList`, `FloatList`, `IntegerList`, `LongList`, `StringList` gibi veri tiplerini tutmakta kullanılmaktadır. Bu liste veri yapıları liste elemanlarını listeye eklendiği sırada tutarlar.

- **ComplexType**

`ComplexType` verilerin isim-değer çiftleri olarak tutulduğu bir tablo şeklindedir. Değerler diğer `ComplexType`'lar dahil olmak üzere herhangi bir tipte olabilir. `ComplexType` soyut bir sınıf olarak tanımlandığından altsınıfları olan `MapType` veya `ModuleType` kullanılmalıdır. `MapType` çalışma zamanında yeni isim-değer çiftlerinin eklenmesine imkan verdiği halde `ModuleType` sadece tasarım zamanında tanımlanan isim-değer çiftlerinin kullanılmasına müsaade eder. `MapType` bastırılırken ya zaten var olan bir özniteliğin değeri değiştirilir ya da yeni bir öznitelik eklenir. Bastırma sırasında var olan bir özniteliğin çıkarılması mümkün değildir. `ModuleType` bastırma sırasında öznitelik eklemeye izin vermemekte ancak önceden tanımlanmış özniteliklerin değerleri değiştirilebilmektedir. `ModuleType` tasarım zamanında tanımlandığından her bir öznitelik için `MapType`'a göre daha fazla kısıtlama konması mümkündür.

2.1.6. Yapılandırılabilir modüller için temel gereksinimler

Daha önce bahsedilen Hertirix'in yapılandırılabilir bütün modülleri `ComplexType` veya bunun alt sınıflarından birinden türetilirler. Geliştirilecek modüller `ComplexType`'ın bir altsınıfı olan ve Heritrix'in bütün modüllerinin de türetildiği `ModuleType` sınıfından türetilmelidir.

Heritrix `ComplexType`'ın nasıl kullanılacağını ve `ComplexType` nesnesi için nasıl kullanıcı arayüzü yaratılabileceğini bilmektedir. Bunun gerçekleşmesi için modülün bazı kurallara uyması zorunludur.

1. Modülün sadece isim parametresi alan bir yapılandırıcısı bulunmalıdır.
2. Yapılandırılabilir olması istenen bütün öznitelikler modülün yapılandırıcısında tanımlanmalıdır.

Bütün modüllerin metin parametresi kabul eden bir yapılandırıcısı bulunmalıdır. Bu metin modülün tanınmasında kullanılmaktadır. Aynı anda sadece bir tane olabilen ve mevcut bir modülün değiştirildiği durumlarda aynı ismin kullanılması önemlidir. Bu gibi durumlarda yapılandırıcı isim parametresini ihmal ederek tasarım zamanında belirlenmiş bir değer kullanabilir. Bu duruma örnek olarak Frontier'ı verebiliriz. Frontier'ın adı her zaman "frontier" olmalıdır. Bu nedenle bütün Frontier'ların gerçekleştirmek zorunda oldukları Frontier arayüzü bir statik isim değişkenine sahiptir.

Aşağıda örnek olarak varsayılan Frontier'ın yapılandırıcısı görülmektedir.

```
...
public static final String ATTR_NAME = "frontier";
public Frontier(String name) {
    //The 'name' of all frontiers should be the same
    //(Frontier.ATTR_NAME)
    //therefore we'll ignore the supplied parameter.
    super(Frontier.ATTR_NAME, "HostQueuesFrontier. Maintains the
internal state of the crawl. ");
...
```

Örnekte de görüldüğü gibi yapılandırıcı temel sınıfın yapılandırıcısını da çağırmalıdır. Bu örnek aynı zamanda bir modülün tanımının da nasıl

yapılacağının göstermektedir. Bu tanım kullanıcı arabirimini tarafından toparlama görevini yapılandırmada kullanıcıya rehberlik etmede kullanılmaktadır.

2.1.7. Özniteliklerin tanımlanması

Modülün yapılandırılabilir olması istenen öznitelikleri modülün yapılandırıcısında tanımlanmalıdır. Bu amaç için `ComplexType`'ın `addElementToDefinition(Type type)` metodu bulunmaktadır. İleride detaylı şekilde açıklanacak ve arama motorumuzun indirilen belgeleri işlemekte kullandığı bir Heritrix modülü olan `LuceneIndexerProcessor` işleyicisinde özniteliklerin nasıl tanımlandığı burada örnek olarak gösterilebilir.

```
…
public class LuceneIndexerProcessor extends Processor implements
CoreAttributeConstants {
    public static final String ATTR_SOLR_USER_ID = "UserId";
    public static final String ATTR_KEYWORDS = "Keywords";

    public LuceneIndexerProcessor(String name) {
        super(name, "LuceneIndexerProcessor processor. " + "A
writer that extracts content of URLs and adds content of each URL
to Lucene index");

        addElementToDefinition(new SimpleType(ATTR_SOLR_USER_ID,
"solr user id", "0"));
        addElementToDefinition(new SimpleType(ATTR_KEYWORDS, "job
keywords", ""));
      }
…
```

2.1.8. Özniteliklere erişilmesi

Özniteliklere erişilirken bastırmalar ve geliştirmelerin dikkate alındığından emin olmak için `getAttribute(String name, CrawlURI curi)` yordamı kullanılmalıdır. Bazen çalışılan bağlam `CrawlURI` tarafından belirlenen kapsamdan farklı olabilir. Bu gibi durumlarda `getAttribute(Object context, String name)` metodu kullanılabilir. Eğer halihazırda bir bağlam yoksa veya global özniteliklere erişilmek isteniliyorsa `getAttribute(String name)` metodu kullanılabilir.

2.1.9. Görevler ve profiller

Heritrix'in görev konsepti toparlama kavramı ile sıkı bir ilişki içindedir. Her bir görev tek bir toparlama içerir. Yani görev Heritrix'te bir toparlamanın uygun şekilde yapılandırılabilmesi için gerekli bütün bilgileri ve ilave olarak toparlamanın durumu hakkında faydalı bilgiler içerir. Aslında profiller de temelde görev ile aynıdırlar ancak yeni görevler yaratmak için şablon olarak kullanılırlar.

Genellikle yeni bir toparlama yaratılırken onu tanımlayan bir profil de yaratılır. İhtiyaç duyulduğunda yapılandırma ve çekirdek URL'ler değiştirilmeden çok sayıda görev yaratılıp çalıştırılabilir. Böylece belirlenen her bir toparlama stratejisi için bir profil oluşturularak yeni bir toparlama görevi yaratılırken şablon olarak kullanılarak zaman kazanılabilir. Görev ve profilleri oluşturan asıl bilgiler diskte XML dosyası olarak tutulmaktadır. Bu dosyaların kullanıcılar tarafından doğrudan düzenlenmeleri mümkün olmakla beraber, kullanıcı arayüzünün kullanılması ortaya çıkabilecek birçok hatayı önleyecektir.

Heritrix birçok görevi aynı anda yönetebilir ancak bu görevlerden sadece bir tanesi aktif yani çalışıyor olabilir. Diğer bütün görevler basit bir kuyrukta sıralarının gelmesini beklerler. Görevler tamamlandıktan sonra görevle ilgili bilgiler daha sonra kullanılmak ve bu görevler esas alınarak yeni görevler yaratabilmek amacıyla saklanırlar.

2.2. Dil Tanıma

Dünya genelinde İnternet erişiminin yaygınlaşması ile birlikte birçok dilde çok büyük miktarlarda metin erişilebilir hale gelmiştir. Bu metinlerin otomatik olarak dilin özelliklerine göre işlenebilmesi, indekslenmesi ve sınıflandırılabilmesi için ön koşul metnin hangi dilde yazıldığının tespit edilmesidir. Örneğin; morfoloji tabanlı kök bulmanın bilgi erişimini etkinliğini artırdığı ve bu işlemin dile özgü algoritmalara ihtiyaç duyduğu bilinmektedir [10]. Dilin tanınması, daha genel olan metnin özniteliklerinin kullanılarak sınıflandırılması probleminin özel bir biçimi olarak görülebilir [11].

Dil tanıma doğal dil işlemenin çeşitli alanlarında kullanılmaktadır. Dil tanımanın muhtemel kullanım alanları aşağıya çıkarılmıştır.

- E-posta filtreleme ve yönlendirme
- Metin madenciliği uygulamaları
- Web sayfalarının dilinin ve sayfa kodlamalarının tespiti
- Bilgi erişim sistemleri
- İçerik tabanlı ve dile özgü web toparlayıcıları ve arama motorları

Verinin ileri düzey morfolojik işleme tabi tutulabilmesi için dil tanıma gereklidir. Köklerin bulunması ve kelimelerin uygun olarak ayrılabilmesi için dokümanın dilinin önceden bilinmesine ihtiyaç duyulmaktadır. Mesela bir imla kontrol programı kullanılacak ise bu programın hangi dile özgü kuralları uygulayacağını bilmesi gereklidir.

Dilin tanınmasında kullanılana tekniklerden bir kaçını kısaca açıkladıktan sonra N-gram metodu ile dilin tanıması konusuna değineceğiz.

2.2.1. Yaygın kelime ve eşsiz harf kombinasyonları

Yaygın kelime yaklaşımının arkasındaki ana fikir her dilin yaygın kullanılan kelimelerinin veritabanında saklanmasıdır. Tanınacak doküman listedeki bütün kelimelerle karşılaştırılır. Bir puanlama sistemi vasıtasıyla o dokümanda geçen kelimelerin en sık karşılaşıldığı liste o dokümanın dilini belirler [12].

2.2.2. İstatistiksel yaklaşım

İstatistiksel dil tanıma konusunda en önemli yayınlarda birisi de Dunning [13] tarafında yayınlanandır. Bu teknikte bir dokümanın hangi dilde yazıldığını tespit etmek için Markov modelleri kullanılmaktadır.

İstatistiksel dil tanımada her dil için örnek metinlerden harf seviyesinde bir dil modeli hazırlanmaktadır. Bu işlem metinleri parçalara ayırarak ve bu parçaları karakter serilerinin görülme sıklığını içeren bir çevrim matrisine koyarak yapılmaktadır.

2.2.3. N-gram yaklaşımı

Dil tanıma konusunda en başarılı yaklaşımlarda birisi de Canvar [14] tarafından ortaya konan N-gram yaklaşımıdır. Dil tanıma için N-gram

kullanmaktaki ana fikir her dilin belirli bazı N-gramları diğerlerinden daha fazla kullanması ve böylece dil hakkında bir ipucu vermesidir.

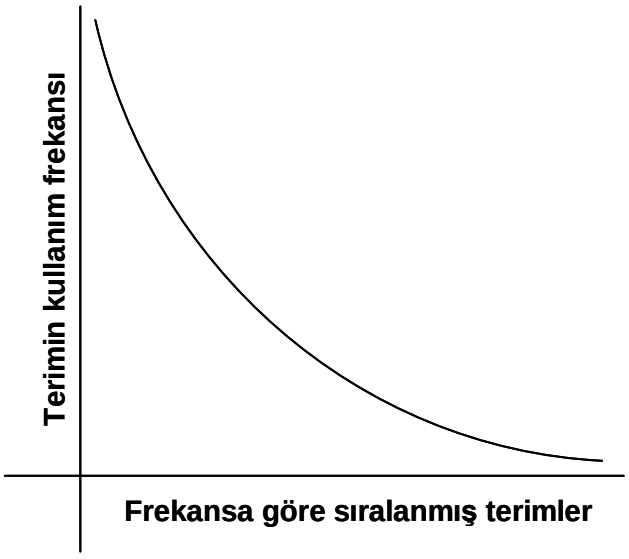
Doğal dillerde her bir kelime farklı frekanslarda olur. Ancak bütün dillerde kelimelerin dağılımı Zipf kanununa uyarlar.

Zipf kanunu: Terimlerin dağılımı ve sıraları arasındaki ilişki sabit bir değere yakınsar.

Eğer *f* kelimenin frekansı ve *r* de kelimenin frekansa göre sıralanmış listedeki sırası ise Zipf kanununa göre:

$$f = \frac{k}{r}$$

Buradan her zaman frekans ve kullanım açısından diğer kelimelere egemen bir gurup kelime olduğunu çıkartılabilir. Kelimeler için Zipf kanunu Şekil 2.4'deki gibi bir grafik ile gösterilebilir. Kelimelerin anlam taşıyan elemanları olan N-gramların frekansı için de Zipf kanunu uygulanabilir.



Şekil 2.4 Zipf kanununa göre kelimelerin dağılımı

N-gramlar kelimelerin alt parçaları olarak görülebilirler. Bu alt parçalar N'in değerine bağlı olarak değişiklik gösterirler. Dil tanıma için tanınacak

dokümanın N-gram profili çıkarılır ve bu profil dile özgü N-gram profili ile karşılaştırır. Tanınacak dokümana mesafesi en yakın olan N-gram profili dokümanın dilini belirler.

Cavnar [14] örnekte görülebileceği gibi bir biri üstene binen N-gramları kullanmıştır. Kelime başlangıcında ve sonunda görülen N-gramların başlangıç veya sonlarına N-gramın bir parçasını oluşturan altçizgi işareti eklenmiştir. Aşağıdaki örnekte "DEFTER" kelimesi için N = 1-4 aralığında muhtemel bütün N-gramlar gösterilmiştir.

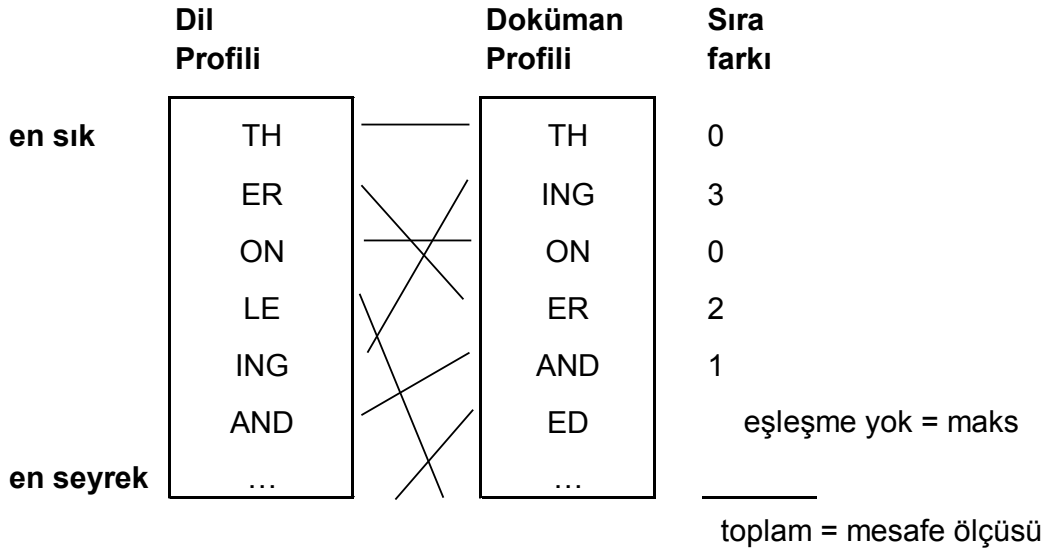
N = 1	D E F T E R
N = 2	_D DE EF FT TE ER R_
N = 3	_DE DEF EFT FTE TER ER_
N = 4	_DEF DEFT EFTE FTER TER_

Algoritma şu şekilde çalışmaktadır. İlk olarak çeşitli dillerdeki metinler teker teker okunmakta ve bütün noktalama işaretleri temizlenmektedir. Her bir kelime başında ve sonunda boşluk olan bir simgeye dönüştürülmektedir. Bütün simgeler işlenerek ve N = 1-4 aralığındaki N-gramlar oluşturulmakta, oluşturulan bu N-gramlar bir `HashTable` veri yapısında saklanmaktadır. Bulunan her bir N-gram eğer Hash tablosunda mevcut değilse tabloya eklenmekte, mevcut ise görülme frekansını gösteren sayaç bir artırılmaktadır. Hash tablosu en sık kullanılan N-gramdan başlayarak aşağıya doğru sıralandığında genellikle uni-gramlar listenin en üst kısımlarında yer alır. Uni-gramlar genellikle dilin alfabetik dağılımını gösterdiğinden kullanılmayabilir. Ancak özellikle Türkçe gibi Latin dillerinden farklı karakterler içeren diller için uni-gramların kullanılması dil tanımlama hassasiyetini artırmaktadır. Bu yöntem her bir dil için tekrarlanır. Oluşturulan N-gram Hash tabloları her dil için N-gram profili teşkil eder.

Bir dokümanın dilini tanıyabilmek için yukarıda bahsedilen prosedürler takip edilerek o doküman için bir N-gram profili oluşturulur. Oluşturulan doküman profili ile mevcut dillerin profilleri arasındaki mesafe karşılaştırılır [14].

Profiller arasındaki mesafe Şekil 2.5'de görülen yöntemle hesaplanır. Test dokümanımızdaki her bir N-gram için karşılaştırdığımız dil profilinde bir karşılık

olabilir. Her iki profilde de sıralaması aynı olan N-gramların arasındaki mesafe sıfır olarak hesaplanır. Eğer N-gramların birbirlerine göre sıralaması farklı ise maksimum mesafe 3 olacak şekilde sıralamaları arsındaki fark hesaplanır. Son olarak N-gramlar arasındaki mesafe farkları toplanır. Bulunan bu rakam karşılaştırılan dil profili ile örnek doküman arasındaki mesafeyi verir. Bu işlem örnek doküman ile karşılaştırılacak bütün dil profilleri tamamlanıncaya kadar tekrarlanır. Mesafesi en küçük olan profil dokümanın dilini belirler.



Şekil 2.5 N-gram profillerinin karşılaştırılması

N-gram yaklaşımının en önemli avantajlarından biriside dile özgü herhangi bir bilgi gerektirmemesidir. Metinlerdeki yazım hataları veya yabancı sözcükler kelimeler N-gramlara bölündüğünden minimal bir sapmaya neden olmakta dilin doğru olarak tanınmasına engel olmamaktadır [14].

Cavnar [14] en sık kullanılan ilk 300 N-gramın metnin yazıldığı dil ile yakından bağlantısı olduğunu, ilk 300 den sonra N-gramların daha çok metnin konusuna göre değişiklik gösterdiğini tespit etmiştir. Bu nedenle dil tanıma amaçlı olarak ilk 300 N-gramın kullanılmasının ve diğerlerinin göz ardı edilebileceği değerlendirilmektedir.

N-gram yaklaşımı kelimelerin sembollere bölünmesine dayandığından Çince, Japoca ve Korece gibi uzak doğu dillerinde kullanılamamaktadır [15]. Bu problemi çözmek için Kikui [16] sembolleştirmeye ihtiyaç duymayan bayt tabanlı

istatistiksel bir metot geliştirmiştir. Bu durum Türkçe için herhangi bir problem teşkil etmemektedir.

2.3. Kök Bulma

Kelimelerin çekim ve yapım eklerinden ayrılarak temel formuna dönüştürülmesine kelimenin kökünün bulunması denilmektedir [17]. Konumuz açısından bulunan kökün morfolojik kök ile aynı olması gerekli değildir. Önemli olan kök gerçekte geçerli olmasa bile birbirleriyle ilişkili kelimelerin aynı köke işaret etmeleridir. Kök bulma işlemi arama motorlarında bilgi erişim etkinliğinin artırılmasında ve diğer doğal dil işleme problemlerinin çözümünde kullanılmaktadır.

Örneğin Türkçe için bir kök bulucu "masadaki", "masada", "masadadır", "masanın" kelimelerinin “masa” kökünden türetildiğini tespit edebilmeli ve bu sözcükleri “masa” köküne indirgemelidir.

Kök bulma konusunda ilk çalışma ve kök bulucu 1968 yılında Julie Beth Lovins tarafından yayınlanmıştır [18,19]. Daha sonra 1980 yılında bir başka kök bulucuda yılında Martin Porter tarafından yayınlanmıştır [20]. Bu kök bulucu çok yaygın olarak kullanılmaya başlanmış ve İngilizce kök bulma için fiili standart haline gelmiştir.

Porter kök bulma algoritmasının birçok uygulaması yazılmış ve dağıtılmıştır. Martin Porter daha sonra çalışmasını daha da genişleterek kök bulma algoritmaları için bir altyapı olan Snowball’u geliştirmiştir [21].

Doğruluğuna, performansına ve karşılaşılan problemleri çözme yöntemine göre farklılık gösteren birçok farklı kök bulma yöntemi vardır. Burada bu yöntemlerden yaygın olarak kullanılan birkaçına değineceğiz.

2.3.1. Kaba kuvvet algoritmaları

Bu algoritmalar kelimelerin kök formları ile çekimli halleri arasında ilişkileri barındıran Çizelge 2.1’de örneği görülen şekle benzer bir sözlük tablosu kullanırlar. Bir kelimenin kökünü bulmak için çekilmiş kelime tabloda aranır ve karşılık gelen kök bulunmaya çalışılır.

Kaba kuvvet algoritmaları kökler ve çekimleri arasındaki ilişkileri tutmak için çok fazla miktarda kaynağa ihtiyaç duyarlar. Türkçedeki kelime sayısı ve dilimizin sondan eklemeli ve çekimli bir dil olması nedeniyle bütün köklerin muhtemel tüm formlarının belirlenmesi be bir sözlük tablosunda tutulması pek gerçekçi değildir.

Bu dezavantajlarına rağmen kaba kuvvet algoritmaları diğer yaklaşımların karşılaştığı bazı problemlerin üstesinden gelebilmektedir [17]. Bir dilde bütün çekimler belirli kuralları takip etmeyebilir. “beyinden” kelimesinin kolaylıkla son eki atılarak “beyin” kökü bulunabiliyorken “beynini” kelimesinin kökünü bulmak basit bir son ek atılması işlemi ile gerçekleştirilemez. Bu tür istisnalar sonek algoritmalarının daha da karmaşık bir hale gelmesine neden olurken kaba kuvvet algoritmalarında bu gibi istisnai durumların çözümü deneyimsiz kullanıcılar için bile genellikle basit bir işlemdir. Yapılması gereken sözlük tablosuna sadece ilave bir kayıt eklemekten ibarettir.

Çizelge 2.1 Kelimeler ve köklerini tutmakta kullanılan sözlük tablosu

Kök	**Kelime**
masal	masalların
masal	masallarına
masal	masallarının
masal	masallarıyla
masal	masallarla
masal	masallı
masal	masalmış
masal	masalsı
masal	masalvari
masa	masam
masa	masamdan
masa	masamdaydı
masa	masamı
Masa	Masamızdaki

2.3.2. Sonek çıkarma algoritmaları

Sonek çıkarma algoritmaları kökler ve çekimleri arasındaki ilişkinin tutulduğu sözlük tablolarına dayanmazlar. Bunun yerine verilen bir kelimenin kök formunu önceden belirlenen sınırlı sayıda kurallara dayanarak tespit etmeye çalışırlar. Aşağıda Türkçe için kullanılabilecek birkaç kural örnek olarak gösterilmiştir.

- Eğer kelime ‘cılık’ ile bitiyorsa ‘cılık’ ekini çıkar dokumacılık → dokuma
- Eğer kelime ‘siz’ ile bitiyorsa ‘siz’ ekini çıkar dikkatsiz → dikkat
- Eğer kelime ‘mek’ ile bitiyorsa ‘mek’ ekini çıkar izlemek → izle

Eğer hedef dil sondan eklemeli bir dilse ve dilin ek ve çekim yapısı sonek çıkarma algoritmaları geliştirilmesine müsaitse sonek çıkarma yaklaşımı etkin olarak kullanılabilir. Sonek çıkarma algoritmaları örneğin ‘ekmek’ gibi yukarıdaki kurala uymayan istisnai durumlarla başa çıkmakta zorlanmaktadırlar.

2.3.3. Lemitizasyon algoritmaları

Bir kelimenin kökünü belirlemede kullanılan daha karmaşık bir yaklaşımda lemitizasyon yöntemidir. Bazı dillerde kelimenin geçtiği yere göre kök bulma kuralları farklılık gösterebildiğinde ilk olarak kelimenin yeri tespit edilir ve farklı kök bulma kuralları uygulanır. Buradaki temel fikir kökü bulunacak kelime hakkında ne kadar fazla bilgiye sahip olursak kökün o kadar doğru bir şekilde bulunabileceğidir.

Bu yöntemin etkinliği kelimenin yerinin ve hangi bağlamda kullanıldığının doğru tespit edilmesine bağlıdır. Kelimenin bağlamının doğru tespit edilmesi zor olduğundan, bu yöntemim sonek çıkarma algoritmalarına karşı üstünlüğü sınırlıdır.

2.3.4. İstatistiksel algoritmalar

İstatistiksel algoritmalar kelimenin kökünü tespit etmede olasılığı kullanırlar. İstatistiksel algoritmalar kökler ve çekilmiş kelimelerin arasındaki

ilişkiler tablosu kullanılarak eğitilirler ve dile uygun bir model oluştururlar. Oluşturulan bu model sonek çıkarma ve lemitizasyonda kullanılanlara benzer karmaşık dilbilgisi kuralları şeklinde ifade edilirler. Diğer algoritmalardan farkı kelimenin kökü bulunurken hangi kuralların hangi sırada uygulanacağına istatistiksel olarak en doğru kökü bulacak şekilde oluşturulan modelden faydalanılarak karar verilmesidir.

2.3.5. Melez yaklaşımlar

Melez yaklaşımlar yukarıda kısaca değinilen iki veya daha fazla yaklaşımı çeşitli şekillerde birlikte kullanan yöntemlerdir. İleride ayrıntılı şekilde açıklanacağı gibi, bu tez kapsamında geliştirilen kök bulma algoritması da melez bir yöntem kullanmaktadır.

Geliştirilen yöntem verilen kelimeyi bir özel isim ve yerleşim birimi adlarının tutulduğu tablolar ile karşılaştırmakta ve bir eşleşme bulursa kelimenin kökünü bulmaya çalışmamaktadır. Eğer kelime bu tablolarda bulunamaz ise, Türkçede yaygın kullanılan köklerin tutulduğu bir tablo ile karşılaştırılarak kelimenin kökü bulunmaya çalışılmaktadır.

3. ÇÖZÜMLER VE UYGULAMA

Bir önceki bölümde arama motorunun alt bileşenleri hakkında ayrıntılı bilgiler verildi. Bu bölümde Türkçe dokümanlar için özelleştirilebilir web tabanlı bir dikey arama motorunun nasıl gerçekleştirilebileceğini ayrıntılı olarak incelenecektir.

Öncelikli olarak arama motorunun genel mimarisinden bahsedilecektir. Daha sonra sırasıyla dokümanların yazıldığı dilin tanınmasından, Türkçe dokümanların analiz edilmesinden, arama motorunun kullanıcı arabiriminden, kullanıcı arabirimi ve toparlayıcının etkileşiminden, dokümanların indekse kaydedilmesi ve sorgulanmasından bahsedilecektir.

3.1. Dikey Arama Motorunun Genel Mimarisi

Bir arama motoru genel olarak aşağıdaki alt bileşenlerden oluşur;

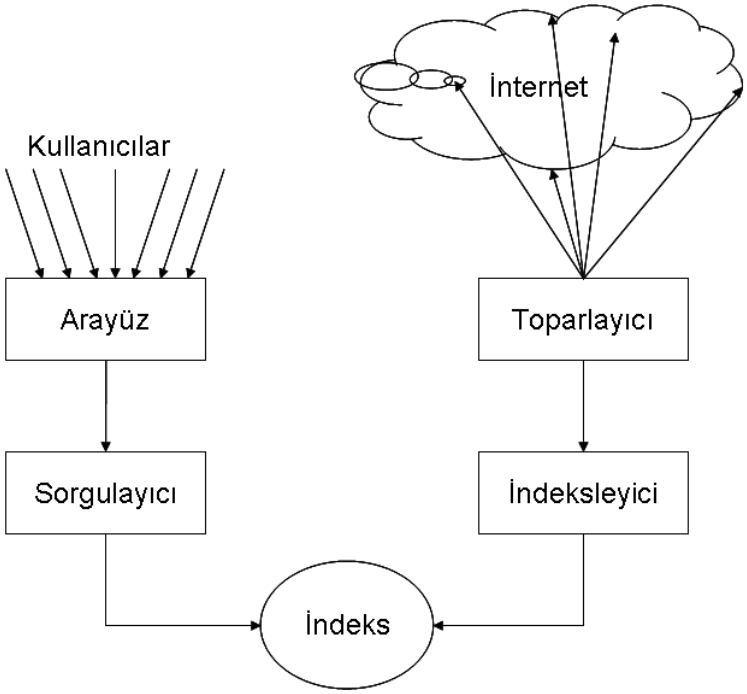
- Toparlayıcı
- İndeksleme sistemi
- Sorgulama sistemi

Tipik bir arama motorunun alt bileşenleri Şekil 3.1'de görüldüğü gibidir.

Ticari olarak hizmet veren arama motorları da mimari olarak temelde benzerdirler. Ticari arama motorlarının temel mimarileri benzer olmakla birlikte alt bileşenlerin tasarımı ve fonksiyonları farklılıklar göstermektedir.

Bizim geliştirdiğimiz dikey arama motoru da temelde yukarıda gösterilen tipik arama motoru mimarisine uymaktadır. Arama motorunun alt bileşenleri motorun tasarımına ve hedeflenen amaca göre değişiklikler gösterebilir.

Geliştirdiğimiz dikey arama motoru toparlayıcı olarak Heritrix'i, indeksleyici ve sorgulayıcı olarak Lucene bilgi erişim kütüphanesini, kullanıcı arayüzü olarak sunucu tarafında Java Server Pages ve Java Servlet teknolojilerini kullanarak geliştirilmiştir. Arama motoru çok kullanıcılı erişime imkan veren web tabanlı bir arayüze sahiptir.



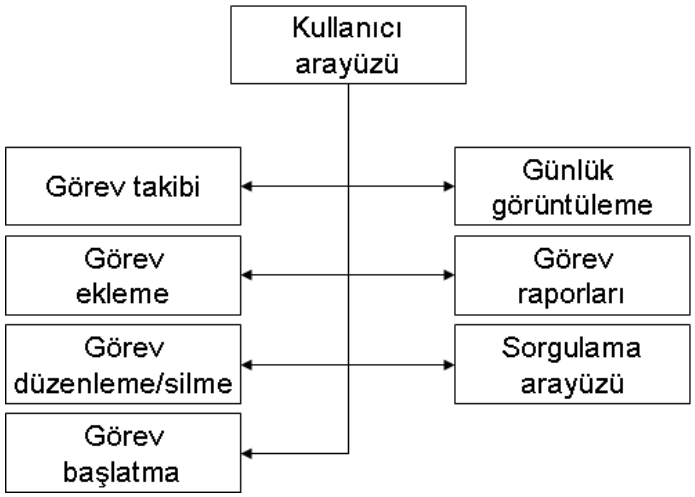
Şekil 3.1 Tipik bir arama motoru

Heritrix toparlayıcısı İnternetin arşivlenmesini amaçlayan ve kar amacı gütmeyen bir kurum olan Internet Archive (IA) tarafından geliştirilmiştir. Tamamen Java programlama diliyle yazılmış açık kaynak kodlu bir projedir. Asıl amacı seçilen web sitelerinin arşivlenmesi amacıyla anlık görünümlerinin kaydedilmesidir. Heritrix toparlayıcısı genişletilebilir bir yapıya sahiptir. Bu nedenle dikey arama motorumuzun toparlayıcı bileşeni olarak tercih edilmiştir.

Lucene yüksek performanslı, ölçeklenebilir bir Bilgi Erişim kütüphanesidir. Apache Jakarta projesinin bir parçası olan Lucene Java diliyle yazılmış, olgun ve ücretsiz açık kaynak kodlu bir projedir. Lucene Apache Yazılım Lisansı çerçevesinde kullanılabilmektedir. Şu anda en popüler ücretsiz Java Bilgi Erişim kütüphanesidir [22].

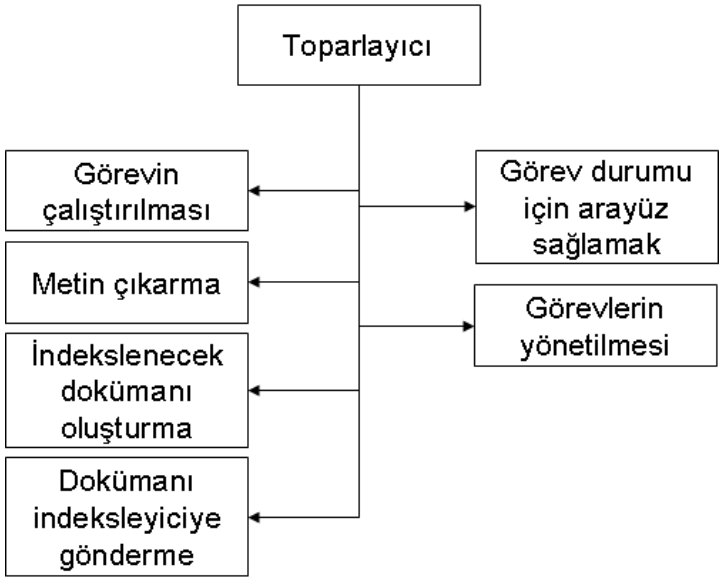
Kullanıcı bilgileri ve oluşturulacak toparlama görevleri ile ilgili bilgileri tutmak amacıyla MySQL veritabanı kullanılmaktadır.

Kullanıcıların arama motoruyla etkileşimi kullanıcı arayüzü vasıtasıyla gerçekleşmektedir. Kullanıcı arayüzünün temel fonksiyonları Şekil 3.2'de görüldüğü gibidir.



Şekil 3.2 Kullanıcı arayüzünün temel fonksiyonları

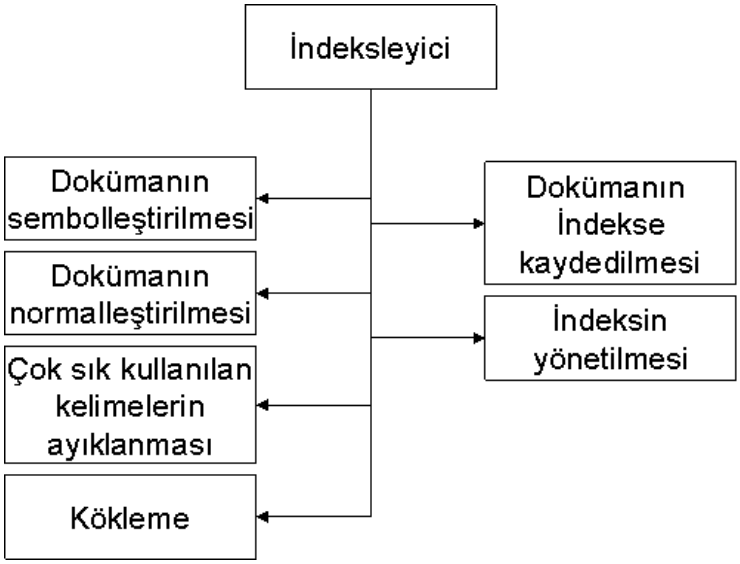
Kullanıcı arayüzü vasıtasıyla oluşturulan toparlama görevi toparlama alt bileşenine verilir ve bu bileşen tarafından çalıştırılır. Toparlayıcı alt bileşeninin genel olarak fonksiyonları Şekil 3.3'de görüldüğü gibidir.



Şekil 3.3 Toparlayıcı alt bileşeninin ana fonksiyonları

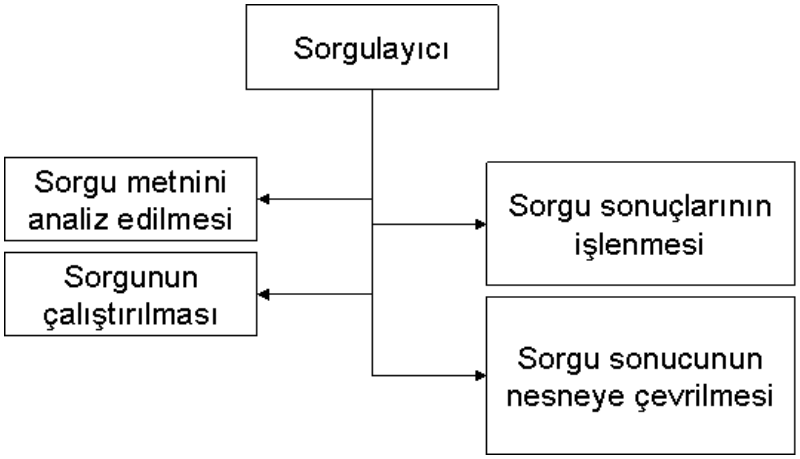
Toparlayıcı tarafından oluşturulan ve indekse eklenecek bir dokümanı temsil eden nesne indeksleyici alt bileşenine gönderilir. İndeksleyici kendisine

gönderilen bu nesneyi işleyerek indekse kaydeder. İndeksleyici bileşeninin temel fonksiyonları Şekil 3.4'de görüldüğü gibidir.



Şekil 3.4 İndeksleyici alt bileşeninin ana fonksiyonları

Yukarıda sayılan bütün işlemlerin ana amacı bilginin kolaylıkla erişilebilir ve kullanılabilir bir hale getirilmesidir. İşte bu amacın yerine getirilmesini sağlayan ana bileşen sorgulayıcıdır. Sorgulayıcı kullanıcı tarafından belirlenen sorgu kriterlerine uyan dokümanların indeksten bulunması ve sunulması görevini yerine getirir. Sorgulayıcı alt bileşenini temel fonksiyonları Şekil 3.5'de görüldüğü gibidir.



Şekil 3.5 Sorgulayıcı alt bileşeninin ana fonksiyonları

Ara motorunun bir bütün olarak çalışabilmesi için bir ortama ihtiyaç vardır. Arama motorumuzu oluşturan ve birbirinden çok farklı görevler yerine getiren alt bileşenlerin tek bir uygulama olarak entegre edilebileceği ortam olarak Java Servlet 2.5 and JavaServer Pages 2.1 standartlarının gerçekleştiren Apache Jakarta projesinin bir ürünü olan açık kaynak kodlu Tomcat uygulama sunucusu kullanılmıştır. Tomcat tam olarak bir uygulama sunucusu değildir. Bizim ihtiyaç duyduğumuz bütün özelliklere sahiptir. Tomcat aynı zamanda temel bir web sunucusu olarak ta görev yapabildiğinden kullanıcı arayüzü dahil olmak üzere bütün bileşenlerin çalışabilmesi için uygun bir ortam sağlamaktadır.

Alt bileşen olarak kullandığımız Heritrix toparlayıcı kütüphanesi ve Lucene bilgi erişim kütüphanesinin çeşitli nesnelerine global bağlamda erişebilmek amacıyla `ServletContextListener` arayüzünü gerçekleştiren `WebappLifecycle` sınıfını kullanmaktadır.

Arama motoru uygulama başlatılırken `WebappLifecycle` sınıfının `contextInitialized(ServletContextEvent sce)` yordamını kullanarak global bağlamda erişilebilecek bir Heritrix nesnesi yaratmaktadır. Aynı yordam içerisinde daha önce herhangi bir indeks klasörü yaratılıp yaratılmadığı kontrol edilmekte eğer indeks klasörü zaten mevcut ise aynı indeks açılarak kullanılmakta mevcut değilse gerekli klasör yaratılmakta ve yeni bir indeks oluşturulmaktadır. `SingletonLuceneWriterSearcher` sınıfının `setLuceneIndexDirectory(File file)` yordamı kullanılarak açılan veya yaratılan indeks klasörü kullanılarak uygulamanın çalışan bütün kanallarının kullanabileceği Lucene `IndexWriter` ve `IndexSearcher` nesneleri yaratılmaktadır. Lucene aynı anda sadece bir `IndexWriter` ve bir `IndexSearcher` nesnesinin kullanılmasına izin verdiğinden her kanal için bu nesneleri yaratmak ve sonlandırmak yerine aynı nesnenin bütün kanallar tarafından kullanılması uygulamanın performansı açısından da zorunludur.

Şimdi arama motorunu oluşturan alt bileşenler sırasıyla ayrıntılı olarak incelenecektir.

3.2. Dokümanların Yazıldığı Dilin Tanınması

Arama motorunda indekslenecek dokümanın dilini tanımak amacıyla N-gram yöntemi kullanılmıştır. N-gram kütüphanesi olarak bir Lucene alt projesi olan Nutch arama motorunda kullanılan `LanguageIdentifierPlugin` eklentisi kullanılmıştır. Ancak bu eklenti Nutch projesine sıkı şekilde entegre edildiğinden doğrudan yazılıma dahil edilerek kullanılamamaktadır. Arama motorumuza entegre edilebilmesi için Nutch yapılandırma altyapısı ve günlük kaydı ile ilgili bölümleri çıkarılmış ve doğru şekilde çalışabilmesi için gerekli değişiklikler yapılmıştır. Bu eklenti varsayılan yapılandırmasıyla Danca, Almanca, Yunanca, İngilizce, İspanyolca, Fince, Fransızca, Macarca, İzlandaca, İtalyanca, Hollandaca, Norveççe, Lehçe, Portekizce, Rusça, İsveççe ve Tayca olmak üzere 17 dili desteklemektedir. Eklentinin Türkçe desteği bulunmamaktadır.

Çizelge 3.1 Türkçe için oluşturulan N = 1-4 aralığındaki ilk 15 N-gram

N = 1		N = 2		N = 3		N = 4	
a	8050375	n_	1759665	lar	492055	ları	239236
e	6059945	e_	1339508	an_	469748	leri	214924
i	5591954	a_	1259154	in_	397159	_bir	212748
n	5064386	ar	1231322	ler	384952	_ve_	190037
r	4432977	i_	1224376	da_	368947	ında	172213
l	4369071	an	1189665	en_	349565	bir_	168525
ı	3083430	la	1128349	eri	344481	erin	161617
k	2997440	_b	1100082	_ka	333800	nda_	158408
d	2886483	er	1042405	_bi	329640	arın	153443
t	2492600	in	968352	_ya	329594	lar_	131715
s	2206695	le	959074	_de	322231	nin_	131525
m	2182586	r_	912900	arı	320409	inde	119715
y	2129503	ı_	898779	de_	317547	dan_	117637
u	1985608	_d	880436	ara	286818	nın_	115642
o	1721853	de	839671	ın_	280171	ını_	115550

Bu eklentiye yeni bir dil desteği eklemek için o dilde yazılmış ve genel olarak dilin yaygın kullanımına paralel metinlerden oluşan bir koleksiyon ile eğitilmesi gerekmektedir. Eğitim için kullanılacak metin bütün noktalama işaretlerinden ve beyaz boşluk olarak bilinen sekme, satır besleme ve yeni satır işaretlerinden arındırılmakta ve her bir kelimenin 1 den N'e kadar N-gramları oluşturulmakta ve bu N-gramların frekanslarıyla ilgili istatistikler yaratılmaktadır. Oluşturulan bu istatistikler 'dilkodu.ngp' adlı dosyaya frekansa göre sıralanmış şekilde her bir satıra "N-gram bir boşluk frekans" olacak şekilde kaydedilmektedir.

Türkçe desteği ekleyebilmek için Milliyet koleksiyonunun [23] ilk 100 bin haberinin <TEXT></TEXT> etiketleri arasında kalan bölümleri bir metin dosyasında birleştirilmiş ve bu dosya kullanılarak Türkçe için N-gram profili oluşturulmuştur. Örnek olarak oluşturulan N-gram profilinin N = 1-4 aralığındaki değerleri için oluşan N-gramların ilk 15 adedi Çizelge 3.1'de görülmektedir.

3.3. Türkçe Dokümanların Analiz Edilmesi

Türkçe, Ural-Altay dil ailesinin Altay koluna bağlı olan ve bitişken dillerden de sondan eklemeli bir dildir. Sözcük türetim ve çekiminde eklerden (yapım ve çekim ekleri) yararlanılır ve çekim ve türetim sırasında sözcük kökleri genellikle herhangi bir değişikliğe uğramaz.

Kelimelerde zamana ve kullanılan alana bağlı olarak sürekli değişmelerin, gelişmelerin olması dilin canlılığının bir göstergesidir. Dil durağan değil, dinamik bir yapıya sahiptir. Dilin söz varlığını oluşturan kelimelerdeki sesler, heceleri ve kelimeleri oluştururken tarihi süreç içerisinde düşerler, yer değiştirirler, türerler, başka seslere benzerler. İşte bütün bunlar, ses olayları başlığı altında incelenir. Dilde ses olayları, çeşitli sebeplerden kaynaklanır. Ancak biz burada konumuz olan kelimelerin köklerinin bulunması açısından önem arz eden orta hece ünlüsünün düşmesi ve kelime sonundaki harflerin yumuşaması olaylarından bahsedilecektir.

- **Orta hece ünlüsünün düşmesi**

Orta hecenin vurgusuz olması sebebiyle, özellikle ğ, r, y, z zayıf ünsüzlerinin yanındaki ünlünün düşmesi olayıdır: ağızı > ağzı, boyunum>

boynum, buradan > burdan, buyuruk > buyruk, dirilik > dirlik, kıvırım > kıvrım, oğulu > oğlu, satılık > satlık, yalınız > yalnız, yanılış > yanlış.

- **Yumuşama**

Kelime sonunda iki ünlü arasında kalan p, ç, t, k sedasız seslerinin sedalılaşarak b, c, d ve g'ye dönmesidir: çorap+ı > çorabı, genç+i > genci, kanat+ı > kanadı, konak+a > konağa.

Tek heceli kelimelerin çoğunda ve sedalılaşma olduğunda anlamı değişecek kelimelerde yumuşama olmaz: atı, haçı, saça, suçu, otu.

3.3.1. Kök bulma ve TurkishPrefixMatchFilter

Türkçe sondan eklemeli bir dil olduğundan arama morumuzda kullandığımız yaklaşım kaba kuvvet ve soneklerin çıkarılması yönteminin bir karışımı sayılabilecek melez bir yöntem kullanılmıştır.

Türkçe metinlerin mümkün olduğunca doğru bir şekilde analiz edilebilmesi için kaynak metni çeşitli işlemlere tabi tutarak indekslenmek üzere hazırlayan `TurkishPrefixMatchAnalyzer` sınıfı geliştirilmiştir. Geliştirilen bu analizci sınıf ilk olarak Lucene `StandardTokenizer` sembolleştirici sınıfını kullanarak metin sembollere ayrılır. Sonraki adımda `TurkishLowerCaseFilter` filtresini kullanarak sembollerin tamamını küçük harfe çevirir. Performansın artırılması ve depolama alanına olan ihtiyacın azaltılması amacıyla Türkçede çok yaygın olarak kullanılan ve sorgu metni içerisinde bulunması çok anlamlı olmayan "ve", "veya", "ile", "ise" gibi hemen hemen bütün dokümanlarda geçen sembolleri ayıklar ve son olarak `TurkishPrefixMatchFilter` filtresini kullanarak Türkçe kelimelerin köklerini bulmaya çalışır.

Türkçe kelimelerin köklerinin bulunması için geliştirilen `TurkishPrefixMatchFilter` sembol filtresi `TurkishPrefixMatchAnalyzer` adlı analizcinin temelini teşkil etmektedir.

Geliştirdiğimiz bu filtre öncelikle bir sembolüm içerisinde "'" kesme işareti olup olmadığa bakmaktadır. Türkçede kesme işareti "Atatürk'ün, İstanbul'un, Aydın'ın" örneklerinde olduğu gibi çoğunlukla özel isimlerin eklerini ayırmakta kullanıldığından sembolüm kesme işaretine kadar olan kesimi özel isim

olarak kabul edilmekte ve ekinden ayrılarak herhangi ilave bir işlemden geçirilmeden indekslenmektedir.

Yani semboller aşağıda gösterilen forma dönüştürülmektedirler.

Atatürk'ün → Atatürk

İstanbul'un → İstanbul

Ahmet'in → Ahmet

Eğer sembol içerisinde kesme işareti yoksa ise bir sonraki adıma geçilmektedir. Bir sonraki adımda sembolün Java `HashTable` veri yapısında tutulan yerleşim birimlerinin isimlerinden herhangi birisiyle örtüşüp örtüşmediğini kontrol etmektedir. Eğer sembol Hash tablosunda mevcut is,e bu sembolü yerleşim birimi adı olarak kabul edilir ve herhangi bir kök bulma işlemine tabi tutmadan ve herhangi bir değişikliğe uğratmadan olduğu gibi indekslenmesine izin verilir. Eğer sembol yerleşim birimleri adlarından herhangi birisiyle eşleşmez ise bir sonraki adıma geçilir.

Yerleşim birimleri isimlerine örnek olarak Çizelge 3.2'deki liste verilebilir. Yerleşim birimlerinin ismi bu aşamada kontrol edilmezse ileriki adımlarda listede görülen yerleşim birimi isimlerinin kökleri bulunacak ve "ağaç" sembolüne dönüştürülecektir. Bu durumda "ağaçoba" kelimesi sorgulandığında aslında ilgili olmamasına rağmen içersinde "ağaç" sembolü geçen bütün dokümanlar döndürülecektir.

Çizelge 3.2 Kökü "ağaç" olan yerleşim birimi adlarına örnekler

Ağaç	**Ağa**
Ağaçlıpınar	Ağadere
Ağaçoba	Ağadibek
Ağaçören	Ağaköy
Ağaçöven	Ağalar
Ağaçpınar	Ağalarobası
Ağaçsaray	Ağapınar
Ağaçseven	Ağayeri
Ağaçsever	Ağadibek
Ağaçyolu	Ağalar

Bu adımda sembol yine Java `HashTable` veri yapısında tutulan ve Türkçede yaygın kullanılan kız ve erkek isimlerinde oluşan bir Hash tablosu ile karşılaştırılmaktadır. Eğer sembol bu tablodaki özel isimlerden herhangi birisi ile eşleşirle, bu sembol özel isim olduğundan ilave bir işleme tabi tutulmadan aynen indekslenmektedir.

Bir sonraki adımda sembol Türkçede yaygın olarak kullanılan isim, sıfat ve fiillerin köklerinde oluşan ve Java `HashTable` veri yapısında tutulan sözcük listesi ile karşılaştırılmaktadır. Bu adımda eğer sembol üç harften fazla ise ilk adımda ilk üç harf alınmakta ve bu sembol Hash tablosunda aranmaktadır. Eğer sembol tabloda bulunursa kaçıncı harfe kadar olan bölümünü tabloda bulunduğu kaydedilmektedir. Bu işlem harf sayısı birer artırılarak kelimenin sonuna kadar tekrarlanmakta ve karşılaşma sağlanan en uzun sembol kök olarak kabul edilmektedir.

Örneğin kelime "delegelerin" olsun. Aşağıdaki tabloyu kullanarak adım adım kelimenin kökü bulunmaya çalışılacaktır. Kelime üç harften uzun olduğu için ilk önce ilk üç harfini alarak kök bulma işlemine başlanır.

	Sembol	**Eşleşme**	**Kaçıncı harf**
1. Adım	del	yok	3
2. Adım	dele	yok	0
3. Adım	deleg	yok	0
4. Adım	delege	var	6
5. Adım	delegel	yok	0
6. Adım	delegele	yok	0
7. Adım	delegeler	yok	0
8. Adım	delegeleri	yok	0
9. Adım	delegelerin	yok	0

Kelime adım adım işlendikten sonra "Kaçıncı harf" sütunundaki en büyük sayıyı buluyoruz ve kelimenin başlangıcından itibaren bu sayı kadar harfin oluşturduğu sembolü kelimenin kökü olarak kabul ediyoruz. Buradaki örnekte en büyük sayı 6 olduğundan "delegelerin" kelimesinin kökü "delege" olarak bulunacaktır.

Eğer tabloda kelimenin kökü olarak kullanılabilecek bir girdi bulunamazsa herhangi bir işlem yapılmadan kelime indekslenmektedir.

Kök bulma işlemimizin son adımını orta hece ünlüsünün düşmesi ve kelime sonundaki “p”, “ç”, “t”, “k” harflerinin yumuşamasından kaynaklanan ve kök bulmada güçlük teşkil eden problemlerin çözülmesidir.

Kelime kökünde harflerin yumuşaması ve ünlü düşmesinden kaynaklanan değişimlerin ortaya çıkardığı farklılaşma kelimenin anlamını değiştirmemekte ancak kökün değişime uğramasına neden olmaktadır. Harf yumuşaması ve ünlü düşmesi nedeniyle Çizelge 3.3’de örnekleri görülen değişimlerden kaynaklanan yeni formlar da kök olarak kabul edilmekte kökler tablosuna eklenmektedir. Böylelikle kelime eklerden dolayı yumuşamış veya ünlü düşmesine maruz kalmış olsa bile kökünün bulunması mümkün olmaktadır. Örneğin “boynunun” sembolünün bir önceki adımda “boyn”, “adağımız” sembolünün kökü “adağ” olarak bulunmaktadır.

Çizelge 3.3 Yumuşama ve ünlü düşmesinden kaynaklanan değişimler

İsim	**Değer**
acz	aciz
adağ	adak
adab	adap
added	addet
aded	adet
aerobiğ	aerobik
aerodinamiğ	aerodinamik
affed	affet
agnostiğ	agnostik
boyn	boyun
beyn	beyin

Bizim kök bulma yaklaşımımız açısında bu kökler suni kökler olarak adlandırılabilir. Çünkü bu köklerin ilave bir işleme tabi tutularak asıl formlarına dönüştürülmeleri gerekmektedir. Bu dönüştürme işlemi bir sonraki adımda bulunan suni kökün Java `HashTable` veri yapısında tutulan yumuşama ve düşme nedeniyle değişime uğrayan kökler tablosunda aranması ve varsa bulunan kökün tabloda hizasında yer alan değer ile değiştirilerek asıl formuna dönüştürülmesi ile sağlanmaktadır. Oluşturulabilecek temsili bir Hash tablosu içeriği Çizelge 3.3’de görüldüğü gibidir.

Kullanılan bu veri yapısı aynı zamanda yaygın kullanım ve yazım hataların düzeltilmesinde hatalı formların hizalarındaki doğru şekle dönüştürülmesinde kullanılabilir. Ayrıca bu tablo aynı anlama gelen birden çok kelimenin tek bir kelime olarak indekslenmesi ve sorgulanmasında da kullanılabilir.

3.4. Kullanıcı Arayüzü

Arama motorumuzun kullanıcı arayüzü, kullanıcıların arama motorunu sorgulayabilecekleri, yeni toparlama görevleri oluşturarak arama motoruna içerik ekleyebilecekleri ve devam eden toparlama görevlerini takip edebilecekleri bir arabirim vazifesini görmektedir.

Bu fonksiyonların İnternete bağlı herhangi bir bilgisayardan kolaylıkla kullanılabilmesini sağlamak amacıyla kullanıcı arabirimi Java Servlet ve Java Server Pages teknolojilerini kullanarak web tabanlı olarak tasarlanmıştır.

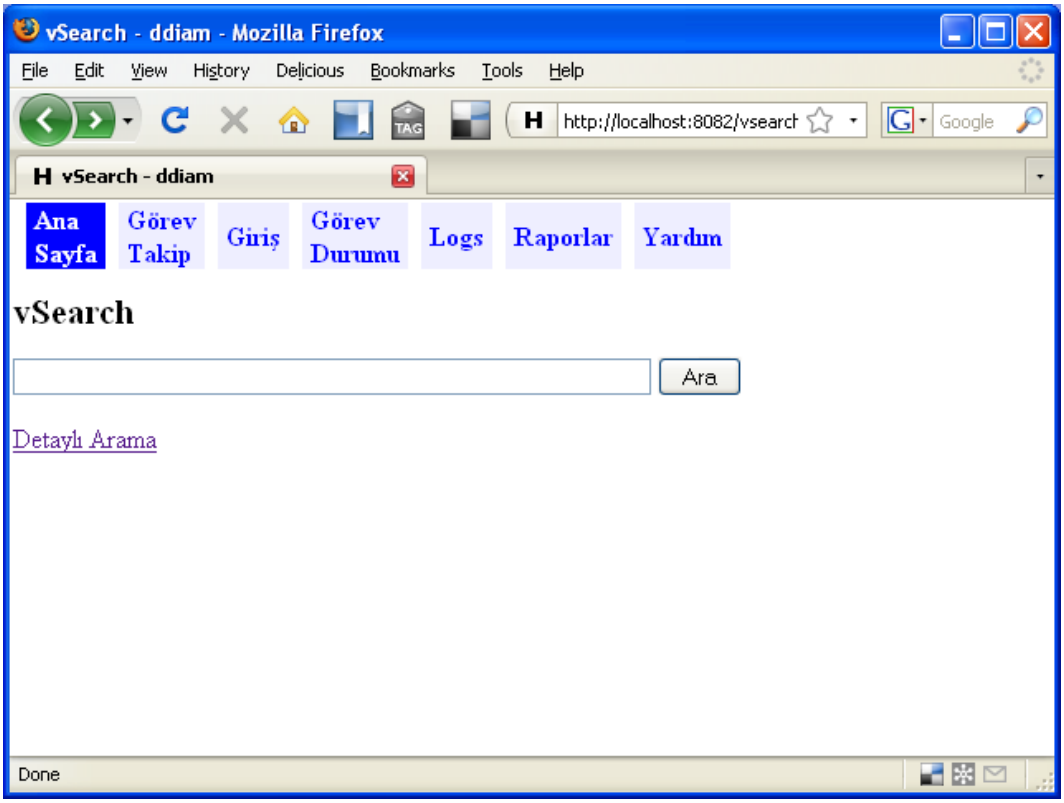
Arama motorumuz yönetici tarafından Şekil 3.6'da şeması görülen MySQL veritabanı tablosunda kayıtlı kullanıcıların web tabanlı arayüzü kullanarak sisteme giriş yapmalarına, yeni bir toparlama görevi oluşturmalarına, mevcut görevleri değiştirebilmelerine, silebilmelerine, bu görevleri çalıştırabilmelerine ve oluşturulan indeksleri sorgulayabilmelerine imkan sağlamaktadır.

Fields | Indexes | Foreign Keys | Triggers | Options | Comment

Name	Type	Length	Decimals	Allow Null	
uid	int	10	0	☐	🔑
display_name	varchar	50	0	☐	
uname	varchar	20	0	☐	
password	varchar	20	0	☐	

Şekil 3.6 Kullanıcı bilgilerinin tutulduğu users tablosu

Şekil 3.7'de Dikey arama motorumuzun ana sayfası görülmektedir.



Şekil 3.7 Dikey arama motorunun ana sayfası

Şimdi adım adım temel birkaç görevin nasıl yerine getirilebileceğini açıklayacağız.

3.4.1. Kullanıcı girişi

Arama motorumuz kayıtlı olmayan kullanıcıların arama motorunu kullanmasına izin vermektedir. Bu kullanıcılar indekste mevcut bütün verileri sorgulayabilmektedirler. Ancak kayıtsız kullanıcıların indekse yeni herhangi bir şey eklemeleri mümkün değildir. Ayrıca kayıtsız kullanıcılar arama motorunun en önemli özelliklerinden birisi olan kişiye özel sorgulama özelliğini kullanamamaktadırlar.

Kayıtlı kullanıcılar sisteme Şekil 3.8'de görülen sayfayı kullanarak giriş yapmaktadırlar. Kullanıcı adı ve şifresi doğrulandıktan sonra sistem kullanıcıyı ana sayfaya yönlendirmektedir.

Şekil 3.8 Kullanıcı giriş sayfası

3.4.2. Yeni toparlama görevi yaratma

Arama motoru ilk defa başlatıldığında indekslenmiş herhangi bir doküman bulunmamaktadır. Sorgulama yapılabilmesi için öncelikle bir toparlama görevi yaratılmalı ve çalıştırılmalıdır. İleride tam olarak toparlamanın ne olduğu ve nasıl çalıştığı detaylı olarak açıklanacaktır. Ancak burada toparlama görevinden kasıt temel olarak indekslenmesi istenen web sitelerinin ve indeksleme kriterlerinin tutulduğu bir veritabanı kaydıdır.

Yeni bir görev eklemek için kullanıcı ana sayfadaki "Yeni Görev" linkine tıklayarak yeni görev ekleme sayfasına ulaşır. Bu sayfada girilmesi gereken bilgiler şunlardır:

- **Görevin Adı**

  Görevi tanımlamak için seçilecek ad yazılır.

- **Çekirdek web sitesi adresleri**

  Bu alana her satıra birer tane olacak şekilde indekslenmesi istenen web sitelerinin adresleri yazılır. Adres sayısında herhangi bir kısıtlama yoktur.

- **İndekslenmesi istenen doküman tipleri**

  Desteklenen doküman türlerinden hangilerinin içeriğinin indeksleneceğini buradan seçilebilmektedir.

- **Dokümanlar içerisinde geçmesi istenen anahtar kelimeler**

  İsteğe bağlı olan bu alana toparlayıcının indirdiği dokümanların içeriğinde bulunması istenen anahtar kelimeler aralarında birer boşluk bırakılarak

yazılır. Toparlayıcı sadece dokümanın içeriğinde bu anahtar kelimelerden bir tanesi var ise o dokümanı indekslemektedir. Örneğin anahtar kelimeler alanına "mayın, pkk, kalleş, terör, iran, ırak" kelimelerini girersek toparlayıcı sadece içerisinde bu kelimelerden herhangi birisi geçen dokümanları indeksleyecek diğerlerini göz ardı edecektir. Böylece indekse sadece ilgilenilen konularla ilgili dokümanlar ekleneceğinden o konuyla ilgili sorgulama sonuçlarının ilgililik oranı artacaktır.

- **Görevin kapsamı**

Yaratılan görev çalıştırılırken girilen çekirdek web adreslerinden itibaren başlayan toparlama görevi herhangi bir sınırlama konmaz ise gittikçe genişleyecektir. Herhangi bir kısıtlama yoksa ve yeterli kaynak ve zaman mevcutsa görev bütün İnterneti kapsayacak şekilde genişleyecektir. Bu nedenle pratikte amaca uygun olarak görevin kapsamına bir sınırlama getirmek gereklidir.

Görevin kapsamı çoktan seçmeli bir alandır. Seçilebilecek değerler ve anlamlarını bir örnekle açıklamak uygun olacaktır. Yaratılan bir görev için aşağıdaki çekirdek adreslerin girdiği varsayılarak her bir seçeneğin ne anlama geldiğini açıklanacaktır.

http://www.milliyet.com
http://haber.mynet.com
http://www.hurriyet.com.tr/saglik

Toparlayıcı çekirdek adreslerden başlayarak sayfaları indirmeye ve indirdiği sayfalardan bağlantıları çıkartmaya başlayacaktır. Toparlayıcının bu üç çekirdek adresten yola çıkarak ilk aşamada Çizelge 3.4'de görülen bağlantıları tespit ettiği varsayılarak her bir seçenek için hangi adreslerin seçilen kapsam içinde kalacağı incelenecektir.

Kapsam: Domain

Domain kapsamı seçilirse Çizelge 3.4'deki 10. satırdaki URL hariç bütün adresler çekirdek olarak girilen alan adlarının birer alt seti olduğundan toparlayıcı bu bağlantıları kapsam içerisinde kabul edecek ve indirmek için planlayacaktır.

Çizelge 3.4 Toparlayıcının tespit ettiği varsayılan URL'ler

Sıra	URL
1	http://www.milliyet.com/haber.asp?haberid=3
2	http://emlak.milliyet.com/emlak/kiralik.asp
3	http://magazin.milliyet.com/guncel.html
4	http://www.mynet.com/depo/magaza/elektonik.jsp
5	http://egitim.mynet.com
6	http://www.mynet.com/guncel
7	http://www.hurriyet.com.tr/saglik/teshis.aspx
8	http://kelebek.hurriyet.com.tr/saglik/cilt.html
9	http://www.hurriyet.com.tr/sondakika.php
10	http://www.vatan.com.tr

Kapsam: Host

Kapsam olarak host seçildiğinde *http://www.milliyet.com* ile *http://emlak.milliyet.com/* adresleri aynı alt alan adının bir parçası olmadığından toparlayıcı sadece 1, 7 ve 9 numaralı adresleri kapsam içersisinde kabul edecektir. Diğer adresler dikkate alınmayacaktır.

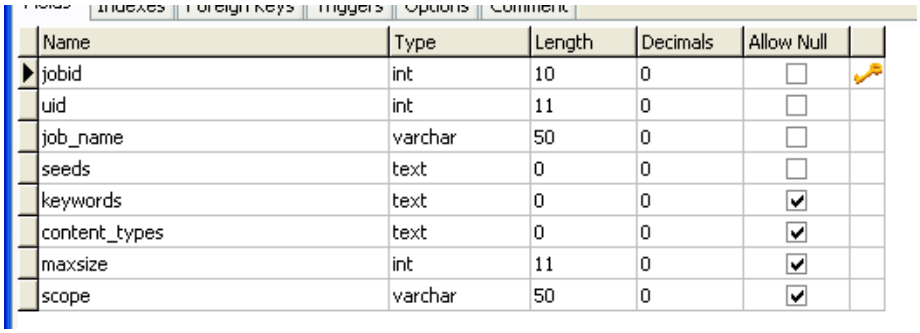
Kapsam: Path

http://www.milliyet.com/saglik/a.asp ile *http://www.milliyet.com/b.asp* adresi aynı dizinde bulunmadığından bu kapsam seçildiğinde yalnızca çekirdek adres ile aynı dizinde bulunan sayfalar dikkate alınır. Yani bu kapsam seçildiğinde sadece 1 ve 8 numaralı adresler kapsam içerisinde kabul edilecektir.

- **Maksimum doküman ebadı**

İnternette zaman zaman çok büyük dosya ebatlarıyla karşılaşmak mümkündür. Bu nedenle ihtiyaca göre dosya ebadı için bir üst sınır koymak gereklidir. Maksimum doküman ebadı bölümüne istenilen değer kilobayt cinsinden girilerek bu değerden daha büyük dokümanların toparlayıcı tarafından indirilmesi engellenebilmektedir.

Yaratılan toparlama görevleri MySQL veritabanında saklanmaktadır. Görevlerin kaydedildiği tablonun şeması Şekil 3.9'da görüldüğü gibidir.



Name	Type	Length	Decimals	Allow Null
jobid	int	10	0	☐
uid	int	11	0	☐
job_name	varchar	50	0	☐
seeds	text	0	0	☐
keywords	text	0	0	☑
content_types	text	0	0	☑
maxsize	int	11	0	☑
scope	varchar	50	0	☑

Şekil 3.9 Toparlama görevleri ile ilgili bilgilerinin tutulduğu crawljob tablosu

3.4.3. Toparlama görevinin güncellenmesi

Kullanıcılar daha önce yarattıkları toparlama görevlerinin herhangi bir parametresini kolaylıkla değiştirebilmektedir. Herhangi bir görevin güncellenebilmesi için "Görev Listesi" sayfasında listelenen görevin hizasındaki "Düzenle" bağlantısına tıklandığında görev düzenleme sayfası açılmaktadır. Böylece kullanıcılara görevin parametrelerini hızla değiştirerek yeniden çalıştırma olanağı da sağlanmış olmaktadır.

3.4.4. Toparlama görevinin silinmesi

Kullanıcılar toparlama görevleri listesini kullanarak diledikleri görevleri silebilmektedirler. Herhangi bir görevin silinmesi eğer görev daha önce çalıştırıldıysa o görevin indekse eklediği dokümanları etkilememektedir.

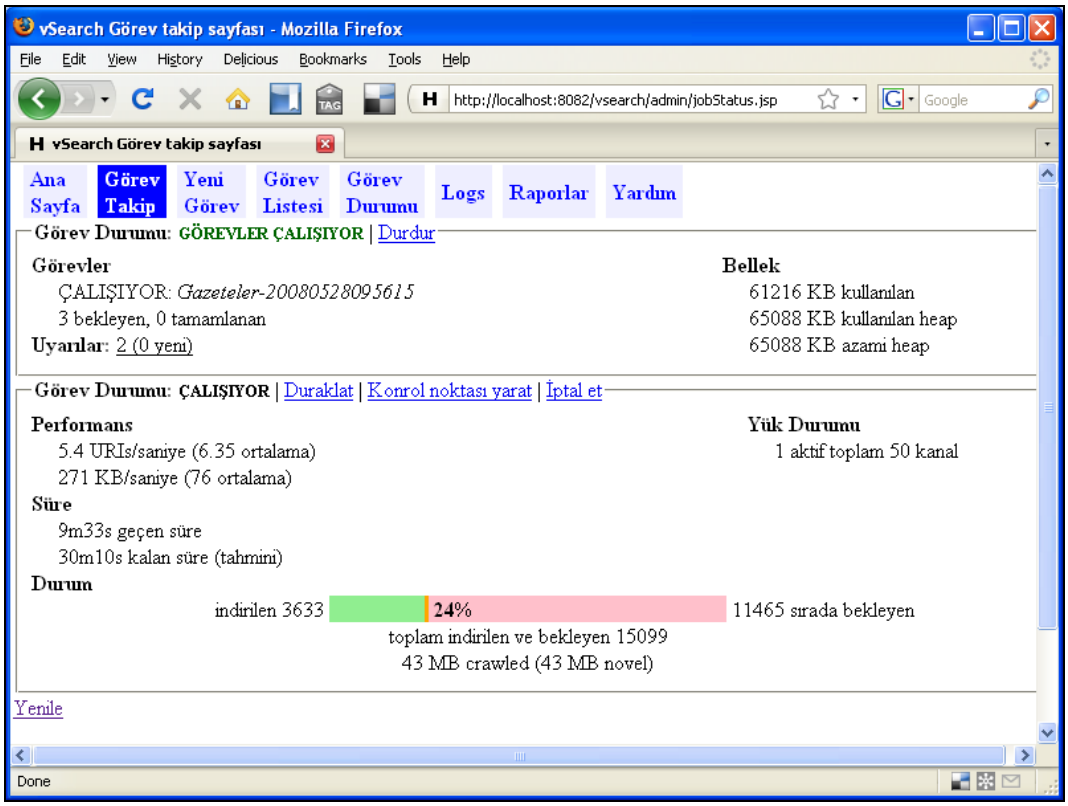
3.4.5. Toparlama görevinin çalıştırılması

Kullanıcılar toparlama görevleri listesini kullanarak istedikleri görevleri çalıştırabilirler. Ancak geliştirilen arama motoru bütün kullanıcılardan gelen görev çalıştırma isteklerini bir basit kuyruğa ekler ve çalışan görev tamamlanınca sıradaki görevden devam eder.

3.4.6. Toparlayıcı durumunun takip edilmesi

Toparlayıcının genel durumuyla ilgili bilgiler "Görev Takip" sayfası kullanılarak izlenebilmektedir. Bu sayfayı kullanılarak çalışan görevi duraklatmak, sürdürmek veya iptal etmek mümkündür. Sayfadan o an kullanılan bellek, azami bellek, çalışan görevin adı, sırada bekleyen görev sayısı, tamamlanan görev sayısı, görevin yürütülmesi sırasında meydana gelen hatalar, saniyede indirilen ortalama sayfa sayısı, ortalama indirme hızı, görevin ne kadar süredir çalıştığı, o ana kadar keşfedilen bağlantı sayısı, indirilen sayfa sayısı ve toplam indirilen veri miktarı gibi bilgileri almak mümkündür.

Durum takip sayfasının ekran görüntüsü Şekil 3.10'da görüldüğü gibidir.



Şekil 3.10 Görev durumu takip sayfası

3.4.7. Raporlar

Devam eden toparlama görevleri ile ilgili daha detaylı bilgiler "Raporlar" sayfasından alınabilmektedir. Bu sayfada görevle ilgili olarak aşağıdaki bilgiler yer almaktadır.

- Toplam keşfedilen URI sayısı
- Sırada Bekleyen URI sayısı
- İndirme işlemi devam eden URI sayısı
- Başarılı olarak indirilen, başarısız olan ve ihmal edilen indirme sayılarının genel toplamı
- Geçerli toparlama görevi için karşılaşılan her bir HTTP durum kodunun toplam sayısı
- Karşılaşılan içerik tipleri ve toplam sayıları ve her bir içerik tipi için indirilen toplam veri miktarları
- Toparlama sırasında karşılaşılan alan adları, her bir alan adı için indirilen doküman sayısı, bu dokümanların toplam büyüklüğü ve en son doküman indirilmesinden o ana kadar geçen süre

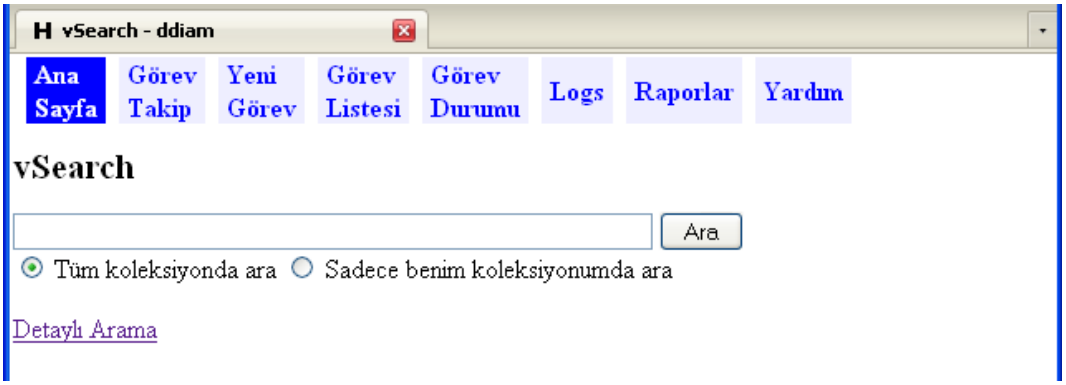
3.4.8. Günlükler

Çalışan toparlama görevi ile ilgili olarak kaydedilen diğer bilgilere "Günlükler" sayfasından erişilebilmektedir. Bu sayfadan aşağıdaki bilgilere erişilebilmektedir.

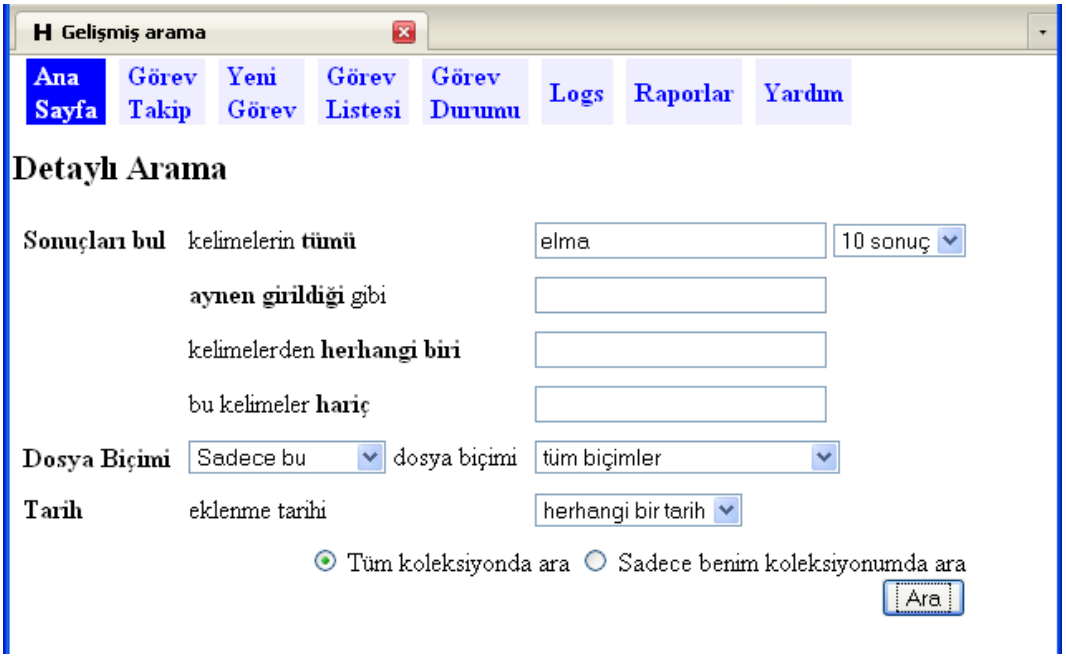
- Toparlama görevi kapsamında indirilen bütün URI'lerin indirilme zamanı, HTTP durum kodu, doküman boyutu, içerik tipi, bağlantı adresinden oluşan günlük kayıtları
- Toparlama görevinin yürütülmesi sırasında oluşan hatalar günlüğü
- İşleyici istatistikleri günlüğü
- Çalışma zamanı hataları günlüğü
- URL hataları günlüğü

3.4.9. Sorgulama

Kullanıcı arayüzü sorgulama için basit ve detaylı olmak üzere iki tür sorgulama imkanı sağlamaktadır. Her iki sorgulama yönteminde de kişiye özel sorgulama imkanı mevcuttur. Basit sorgulama sayfası Şekil 3.11'de, detaylı sorgulama sayfası Şekil 3.12'de görüldüğü gibidir.



Şekil 3.11 Basit sorgulama sayfası



Şekil 3.12 Detaylı sorgulama sayfası

Detaylı sorgulama sayfası ile yapılabilecek bütün sorgular aslında basit sorgulama sayfasını kullanarak da yapılabilmektedir. Ancak kullanıcıların Lucene sorgu işleyicisinin nasıl çalıştığını detaylı olarak bilmeleri gereklidir. Kullanıcıların gereksiz detaylarla uğraşmalarını engellemek maksadıyla detaylı sorgu sayfası hazırlanmıştır.

Dokümanlar içerisinde mutlaka olması istenen terimler, aynen girildiği gibi geçmesi istene ifadeler, doküman içeriğinde girilen terimler listesinden herhangi birisinin geçmesi istenenler, doküman içeriğinde bulunmaması istenen

terimler belirlenebilmektedir. Bunlara ilave olarak sorgulanan ifadenin hangi doküman türü içerisinde aranması gerektiği ve dokümanın eklendiği tarih de koşul olarak belirlenebilmektedir.

3.5. Toparlayıcı

Daha öncede bahsettiği gibi toparlayıcı olarak genişletilebilir bir mimariye sahip olan Heritrix kullanılmaktadır. Toparlayıcının mimarisini, yapılandırma altyapısını ve genel olarak nasıl çalıştığını daha önce açıkladığından burada sadece dikey arama motorunda toparlayıcının nasıl kullanıldığı açıklanacaktır.

Konunun detaylarına girilmeden önce toparlama görevinin çalıştırılması açısından kullanıcı arayüzü ile toparlayıcının etkileşiminin nasıl sağlandığı detaylı olarak açıklanacaktır.

3.5.1. Kullanıcı arayüzü ve toparlayıcının etkileşimi

Daha önce belirtildiği gibi kullanıcılar ve görevlerle ilgili bilgiler MySQL veritabanında tutulmaktadır. Kullanıcı arabirimi kullanıcılar ve görevlerle ilgili işlemleri bu bilgileri kolaylıkla yönetmek için geliştirilen `DbUtils` sınıfını kullanarak gerçekleştirmektedirler. Burada yapılan işlem temel olarak bu bilgilerin veritabanına yazılması ve okunmasından ibarettir. Ancak kullanıcı arayüzü yardımıyla yaratılan ve veritabanında bilgileri kayıtlı bir toparlama görevlerinin çalıştırması farklılık göstermektedir.

Kullanıcılar arayüzü kullanarak herhangi bir toparlama görevini çalıştırdıklarında sırasıyla aşağıda açıklanan işlemler gerçekleştirilmektedir.

1. Toparlama görevi ile ilgili bilgiler `DbUtils` sınıfının `getJobInfo(int jobId)` yordamı kullanılarak veritabanından alınır.
2. Örnek bir toparlama görevini temsil eden ve şablon olarak kullanılan `template.xml` dosyası diskten okunur. Bu dosyada "#" karakterleri arasında belirtilen yapılandırılabilir bölümler veritabanından alınan toparlama görevinin bilgileriyle değiştirilir. Bir görevin yaratılması için şablon olarak kullanılan görev dosyasında değiştirilmesi gereken parametreler Şekil 3.13'de görüldüğü gibidir.

3. Toparlayıcının görevlerle ilgili bilgileri ve kayıtları tutmakta kullandığı web uygulamasının ana dizininde yer alan `crawl-jobs` klasörü içerisine ilgili görev için yeni bir klasör yaratılır. Bu klasörün adı; görevin adına ilave edilen yıl, ay, gün, saat, dakika ve saniye bilgilerinden oluşur.
4. Oluşturulan görev dosyası bu klasöre `order.xml` adıyla kaydedilir. Görevin çekirdek URL'leri her satırda bir URL olacak şekilde aynı klasörün içersine `seeds.txt` adıyla kaydedilir.
5. Oluşturulan `order.xml` ve `seeds.txt` dosyalarındaki bilgiler Heritrix'in `addCrawlJob(String orderPathOrUrl, String name, String description, String seeds)` yordamına parametre olarak gönderilir ve toparlama görevinin yaratılması sağlanır.

```
<name>#job.name#</name>
<date>#job.date#</date>
<long name="max-length-bytes">#document.maxsize#</long>
<string name="regexp">#contentype.regexp#</string>
<string name="UserId">#user.id#</string>
<string name="Keywords">#keywords#</string>
```

Şekil 3.13 Görev dosyasının yapılandırılabilir parametreleri

3.5.2. Toparlayıcının çalışması

Heritrix varsayılan yapılandırmasıyla web sitelerinin anlık görünümlerini arşivlemek için kullanıldığından geliştirilen dikey arama motorunun ihtiyaç duyduğu fonksiyonları yerine getirememektedir. Bu nedenle Heritrix arama motorunun ihtiyaçlarını karşılayacak şekilde genişletilmiştir.

Yapılan işlem temel olarak Heritrix'in sadece istenilen içerik tiplerini indirecek şekilde yapılandırılması ve indirilen dokümanları belirlenen kurallara göre işleyerek indeksleyen bir işleyici modül yazılmasıdır.

Geliştirilen bu işleyiciye `LuceneIndexerProcessor` adı verilmiştir. Burada işleyicinin nasıl çalıştığını detaylı olarak anlatılacaktır.

Bütün işleyiciler gibi `LuceneIndexerProcessor` da `org.archive.crawler.framework.Processor` sınıfını genişletmektedir. `Processor` sınıfının `innerProcess(CrawlURI)` yordamı bastırılmakta ve bu

yordama bir parametre olarak gönderilen `CrawlURI` nesnesi kullanılarak ihtiyaç duyulan işlemler gerçekleştirilmektedir.

İşleyicinin yapılandırılabilir olması istenen özniteliklerinin işleyicinin yapılandırıcısında tanımlanması gerekmektedir. Bu nedenle geliştirilen işleyicide yapılandırılabilir olmasını istenilen özniteliklere diğer sınıfların da erişebilmesini sağlamak için öznitelikler statik sınıf değişkeni olarak tanımlanmaktadır.

`LuceneIndexerProcessor` iki adet yapılandırılabilir öznitelik tanımlamaktadır. Birincisi toparlama görevini yaratan ve çalıştıran kullanıcıyı belirlemekte kullanılan `UserId` ikincisi indekslenecek dokümanların içeriğinde olması istenen anahtar kelimeleri belirleyen `Keywords` özniteliklerdir.

Bu sınıf değişkenleri `SimpleType` yapılandırma nesnesi olarak `LuceneIndexerProcessor` işleyicisinin yapılandırıcı yordamı kullanılarak Şekil 3.14'de görüldüğü gibi varsayılan değerleri ile yapılandırma altyapısına eklenmektedir.

```
public class LuceneIndexerProcessor extends Processor
    implements CoreAttributeConstants {

    public static final String ATTR_SOLR_USER_ID = "UserId";
    public static final String ATTR_KEYWORDS = "Keywords";

    public LuceneIndexerProcessor(String name) {
        super(name, "LuceneIndexerProcessor processor. " +
  "A writer that adds each URL content to Lucene
        index");

        addElementToDefinition(new
        SimpleType(ATTR_SOLR_USER_ID, "solr user id", "0"));

        addElementToDefinition(new SimpleType(ATTR_KEYWORDS,
        "job keywords", ""));
    }
...
```

Şekil 3.14 LuceneIndexerProcessor işleyicisinin varsayılan yapılandırması

Bu öznitelikler yeni bir toparlama görevi başlatılırken toparlama görevini temsil eden XML formatındaki `order.xml` görev dosyasından okunmaktadır. Bu

görev dosyasında tanımlanmayan öznitelik değerleri için sınıfın yapılandırıcı metodunda belirttiğimiz varsayılan değerler kullanılmaktadır.

Yapılandırılabilir olarak belirlenen özniteliklerden `ATTR_SOLR_USER_ID` özniteliği o anda çalışan görevin hangi kullanıcı tarafından yaratılıp çalıştırıldığının belirlemektedir. Dokümanlardan çıkarılan metin indekslenirken kullanıcı bilgisi de indekse eklenmekte böylece kişiselleştirilmiş koleksiyonlar oluşturmak mümkün olmaktadır. Diğer bir öznitelik olan `ATTR_KEYWORDS` ise kullanıcının dokümanın içeriğinde olmasını istediği anahtar kelimeleri tutmakta kullanılmaktadır. Bu özniteliğin değeri indekslenecek dokümanlar içerisinde mutlaka bulunması istenen anahtar kelimeleri belirler.

`LuceneIndexerProcessor` işleyicisi kendisine gönderilen her bir `CrawlURI` nesnesi için `innerProcess(CrawlURI)` yordamını çalıştırır. Bu yordama ilave olarak kendisine verilen `CrawlURI` nesnesini kullanarak dokümanları işleyen ve indeksleyen `indexDocument(CrawlURI curi)` yordamı ve dokümanlardan çıkarılan metin içerisinde istenen anahtar kelimelerin bulunup bulunmadığını test eden `hasKeywords(String content, String keywords)` yordamı mevcuttur.

Burada adım adım `LuceneIndexerProcessor` işleyicisinin nasıl çalıştığını açıklanacaktır.

`CrawlURI`'nin `is2XXSuccess()` yordamını kullanarak HTTP işlemlerinin başarıyla sonuçlanıp sonuçlanmadığı kontrol edilmektedir. Eğer HTTP cevap kodu 2xx aralığında değilse URI'nin işlemesinden vazgeçilmektedir.

Sonraki adımda Heritrix şu anda sadece HTTP ve HTTPS protokollerini desteklediğinden eğer URI bu protokollerden birisini kullanmıyorsa işleme son verilmektedir.

Daha sonra `CrawlURI`'nin `HttpRecorder` nesnesine erişerek kaydedilen akış uzunluğu kontrol edilmektedir. Akış uzunluğu sıfıra eşitse yani herhangi bir bilgi yoksa işleme son verilmektedir.

Eğer yukarıdaki adımlar başarıyla geçildiyse `CrawlURI` nesnesi indekslenmek üzere `indexDocument(CrawlURI curi)` yordamına gönderilmektedir. Bu yordam indekslenecek her bir doküman için Çizelge 3.5'de görülen alan bilgilerini oluşturur.

Çizelge 3.5 Dokümanlar indekslenirken kaydedilen alanlar.

Alan adı	**Açıklama**
uid	Toparlama görevinin sahibini belirten kullanıcı numarası
url	İndekslenen sayfanın adresi
ctype	Dokümanın içerik tipi
title	Dokümanın başlığı
content	Dokümanın içeriğinin tamamı
host	Dokümanın ait olduğu alan adı
tstamp	Dokümanın işlendiği tarih zaman bilgisi
language	Dokümanın içeriğinin dili

Tabloda görülen alanların nasıl oluşturulduğunu aşağıda açıklanmıştır.

uid **:** Görev dosyasında bulunan "UserId" parametresine Heritrix yapılandırma altyapısının getAttribute("UserId") yordamı kullanılarak ulaşılmaktadır.

url **:** CrawlURI nesnesinin curi.getUURI().getHostBasename() yordamı kullanılarak elde edilmektedir.

ctype **:** CrawlURI nesnesinin getContentType() yordamı dokümanın içerik tipini döndürmektedir.

title **:** Doküman türünü işleyen metin çıkarıcı sınıfının getTitle() yordamı çağrılarak elde edilmektedir.

content **:** Dokümanın içeriğinin metin çıkarıcılar tarafından metne dönüştürülmüş halidir. İlgili metin çıkarıcı sınıfın getContent() yordamı çağrılarak elde edilmektedir.

host **:** `CrawlURI` nesnesinin `curi.getUURI().getHostBasename()` yordamı çağrılarak URI'nin alan adı elde edilebilmektedir. Örneğin URL adresi *http://www.milliyet.com.tr/2008/06/01/index.html* olan sayfanın alan adı *milliyet.com.tr* olarak döndürülür.

tstamp **:** Dokümanın işlendiği ve indekslendiği tarih yıl ay gün formunda kaydedilir. Örneğin 31 Aralık 2008 tarihi için 20081231 olarak kaydedilir.

language **:** İşleyici tarafından `LanguageIdentifier` sınıfının `identify(String text)` yordamı çağrılarak doküman dilini tanımlayan iki karakterden oluşan dil kodu elde edilmektedir.

Burada en önemli konu doküman içeriğinin indekslenmesidir. Aşağıda doküman içeriğinin elde edilmesi ve indekslenmesi adım adım açıklanmıştır.

`CrawlURI` nesnesinin `getContentType()` yordamı kullanılarak dokümanın içerik tipi belirlendikten sonra içerik türüne uygun metin çıkarıcı sınıfın `InputStream` parametresi kabul eden yapılandırıcı metodu kullanılarak yeni bir metin çıkarıcı nesne yaratılır. Yaratılan bu nesnenin `getTitle()` metodu kullanılarak dokümanın başlığı döndürülür. Eğer dokümanın başlığı yoksa null değeri döndürülür. Dokümanın gövde metni ise aynı nesnenin `getContent()` yordamı çağrılarak elde edilir.

Eğer dokümanın başlığı yoksa `null` değeri "Başlıksız" olarak değiştirilir.

Dokümanın gövde metni `null` ise veya yazdırılamayan karakterler temizlendikten sonda uzunluğu sıfıra eşitse bu doküman indeksleme işlemine tabi tutulmaz ve atlanır.

Toparlama görevi oluşturulurken dokümanların içeriğinde olması istenen anahtar kelimeler alanı boş bırakılmamışsa dokümandan çıkarılan metin ve anahtar kelimeler `hasKeywords(String content, String keywords)` yordamına gönderilir. Bu yordam metin içerisinde anahtar kelimelerden en az biri varsa `doğru` yoksa `yanlış` değeri döndürür. Dönen değer `doğru` ise doküman indekslenir değilse indekslenmeden atlanır.

İndekslenecek her bir doküman için bir `VSearchDocument` nesnesi yaratılır. Bu nesnenin Şekil 3.15'de görülen yordamları kullanılarak doküman bilgileri bu nesneye eklenir.

```
setUid(String uid)
setUrl(String url)
setCtype(String ctype)
setTitle(String title)
setContent(String content)
setHost(String host)
setTstamp(String tstamp)
setLanguage(String language)
```

Şekil 3.15: VSearchDocument nesnesinin yordamları

Oluşturulan `VSearchDocument` nesnesinin indekslenmesi için bir `LuceneIndexer` nesnesi yaratılır ve bu nesnenin `indexDocument(VSearchDocument doc)` yordamı kullanılarak dokümanı temsil eden nesnenin indekslenmesi sağlanır.

3.6. İndeksleyici

Toparlayıcı alt bileşeni tarafından yaratılan `VSearchDocument` nesnesi `LuceneIndexer` sınıfı tarafından indekslenmektedir. `LuceneIndexer` nesnesi `indexDocument(VSearchDocument doc)` yordamına parametre olarak gönderilen dokümanının alanlarını kullanarak bir `org.apache.lucene.document.Document` nesnesi yaratır. Bu nesneyi `SingletonLuceneWriterSearcher` nesnesinin `getLuceneIndexWriter()` yordamını kullanarak referansını elde ettiği `IndexWriter` nesnesinin `updateDocument(Term, Document)` yordamını kullanarak indekse kaydeder. Bu yordama `Term` parametresi olarak dokümanın `url` özniteliği gönderilir. Yordam `url` parametresini indekste arar. Eğer mevcutsa ilgili dokümanı günceller, değilse yeni bir doküman olarak ekler.

Bilindiği gibi Lucene kütüphanesi bir doküman indekse kaydedilirken dokümanın alanlarının isteğe bağlı olarak işlenmesine imkan tanımaktadır. Arama motorunda dokümanın bazı alanlarının hem analiz edilmesi hem indekslenmesi hem de daha sonra Sorgulayıcı tarafından gösterilebilmesi için asıllarının indekse kaydedilmesi gerekmektedir.

Dokümanın indekse kaydedilen alanları ve bu alanların kayıt özellikleri Çizelge 3.6'da görüldüğü gibidir.

Çizelge 3.6 Dokümanın indekse kaydedilen alanlar ve bu alanların kayıt özellikleri

Alan Adı	**Analiz Ediliyor**	**İndeksleniyor**	**Kaydediliyor**
uid	Hayır	Evet	Evet
url	Hayır	Evet	Evet
ctype	Hayır	Evet	Evet
title	Evet	Evet	Evet
content	Evet	Evet	Evet
host	Hayır	Evet	Evet
tstamp	Hayır	Evet	Evet
language	Hayır	Evet	Evet

İndekse kaydedilen dokümanın analiz edilen alanları özellikle Türkçe dokümanları analiz etmek için geliştirilen `TurkishPrefixMatchAnalyzer` analizcisi kullanılarak analiz edildikten sonra indekse eklenmektedir.

Dokümanın başlığı ve içeriğinden çıkarılan metin birleştirilerek `content` alanına birlikte eklenmektedir. Böylelikle sorgulama sırasında doküman başlığı ya da içeriğinde aranan ifadenin geçmesi durumunda dokümanı bulmak mümkün olabilmektedir.

3.7. Sorgulayıcı

Kullanıcılar tarafından girilen sorgular bu alt bileşen tarafından işlenmektedir. Temel olarak görevi kullanıcının girdiği sorgunun Lucene tarafından anlaşılacak bir formata dönüştürülmesi ve sorgu sonuçlarının işlenmesidir.

Sorgulamanın doğru şekilde çalışabilmesi için dokümanlar indekse kaydedilirken nasıl gerekli alanları analiz ediliyorsa, sorgulama sırasında da sorgu ifadesinin aynı şekilde tercihen aynı analizci kullanılarak işlenmesi gereklidir. Arama motoru sorguları analiz ederken indekslemede de kullandığı `TurkishPrefixMatchAnalyzer` analizcisini kullanmaktadır.

Lucene kütüphanesinin `org.apache.lucene.queryParser.QueryParser` sorgu işleyici sınıfı kullanıldığında sorgu metninin analiz edilmesi basit sorgularda herhangi bir probleme neden olmasa da detaylı sorgularda ve Şekil 3.16'da görüldüğü gibi sorgu sonuçlarının belirli bir alan adına göre filtrelendiği durumlarda problemlere neden olabilmektedir.

Şekil 3.16 Sorgu sonuçlarının alan adına göre filtrelenmesi

Sorgu parametresi olarak kullanılabilecek alanlardan uid, ctype, tstamp ve language alanları `TurkishPrefixMatchAnalyzer` analizcisi tarafından analiz edildiğinde herhangi bir problem teşkil etmemektedir. Ancak host alanı `TurkishPrefixMatchAnalyzer` tarafından analiz edildiğinde bu alanın kökü bulunmaya çalışıldığından beklenmedik sonuçlara neden olmaktadır.

Bu problemin çözümü için `QueryParser` sınıfını genişleten `VSearchQueryParser` sınıfı geliştirilmiştir. Bu sınıf sorgu içerisinde bulunan host alanının analiz edilmesini engelleyerek sorunu ortadan kaldırmaktadır. Şekil 3.16'da görülen sorgu QueryParser sınıfı ile analiz edildiğinde sorgu "host:saglik.milliyet content:elma" şeklinde yorumlanmaktadır ki bu durum ya sonuçların hatalı olmasına ya da sonuç döndürülememesine neden olacaktır.

Sorgu ifadesi analiz edildikten sonra Lucene `IndexSearcher` sınıfının `searcher.search(query)` yordamına parametre olarak gönderilir ve sorgu çalıştırılır.

Arama motorunda yukarıda açıklanan işlemleri basitleştirmek ve sorgulama alt bileşeni için bir arabirim oluşturmak amacıyla `LuceneSearcher` sınıfı geliştirilmiştir. Bu sınıfın `LuceneSearcher(IndexSearcher searcher, Analyzer analyzer, String q, int start, int rows)` yapılandırıcı yordamı kullanılarak sorgulama işlemi gerçekleştirilebilir. Bu yordama sorguyu çalıştıracak `IndexSearcher` nesnesi, sorgu analiz edilirken kullanılacak analiz sınıfı, sorgu ifadesi, sorgu sonuçlarının hangi dokümandan itibaren işleneceği ve kaç adet sonucun döndürüleceği parametre olarak gönderilir.

`LuceneSearcher` nesnesi yapılandırıcı yordamında gönderilen parametrelere uygun olarak indeksi sorgular ve sorgu sonucunda bulunan dokümanlardan birer `VSearchDocument` nesnesi yaratır.

Oluşturulan `VSearchDocument` nesneleri sorgu içerisinde geçen ifadelerin sorgu sonucunda vurgulanmış olarak gösterilebilmesi için `highLite(VSearchDocument v)` yordamı kullanılarak tekrar işlenir. Başlık uzunluğu belirlene değerden fazla olan dokümanların başlıkları kırpılır.

Sorgu ifadelerinin vurgulanması için bir Lucene eklentisi olan `highlighter` kullanılmaktadır. Bu eklenti sorgu ifadelerinin istenilen metin içerisinde işaretlenmesi ve metinden istenilen uzunlukta ve içerisinde mümkün olduğu kadar fazla sorgu ifadesi bulunacak şekilde bir parça çıkarılması için kullanılmaktadır. Biz sorgu ifadelerinin metin içerisinde vurgulanması ve web sayfasında vurgulu bir şekilde gösterilebilmesi için başlangıç ve bitiş için etiketleri kullanılmıştır.

Şekil 3.16'da sorgulama için kullanılan "elma" anahtar kelimesinin dokümanın içerisinde bulunup nasıl vurgulandığı görülebilir.

4. DENEYLER VE SONUÇLAR

Bu bölümde arama motorunun ve arama motorunu oluşturan alt bileşenlerin etkinliği ve performansını ölçmek ve değerlendirmek amacıyla gerçekleştirilen testler yer almaktadır.

4.1. N-gram Yaklaşımı Kullanarak Türkçe Dil Tanıma Testi

Daha önce arama motorunda bir dokümanın dilini tanımak için N-gram yaklaşımı kullandığı belirtilmiştir. Burada N-gram yaklaşımının dil tanıma konusundaki etkinliği incelenecektir.

Dil tanıma için kullanılan `LanguageIdentifier` eklentisinin Türkçe, İngilizce, Almanca, Fransızca ve İtalyanca dil tanıma etkinliğini tespit etmek amacıyla her dilden seçilen 650 adet doküman kullanılarak bir test yapılmıştır. Türkçe testi için Milliyet koleksiyonundan [23] metin kısmının uzunluğu 2048 karakterden uzun olan haberlerden seçilen 650 adet haber kullanılmıştır. İngilizce, Almanca, Fransızca ve İtalyanca için Avrupa Parlamentosunun 1996-2003 yılları arasındaki tutanaklarının paralel çevirisinden oluşturulan koleksiyonun üçüncü versiyonu kullanılmıştır [24].

Test maksadıyla her bir dil için o dilde yazılmış dokümanlardan sırasıyla 8, 16, 32, 64, 128, 256, 512, 1024 ve 2048'er karakter alınmış ve alınan metin parçalarının dili tanınmaya çalışılmıştır. Test sonuçları Çizelge 4.1'de ve grafiksel olarak Şekil 4.1'de görüldüğü gibidir.

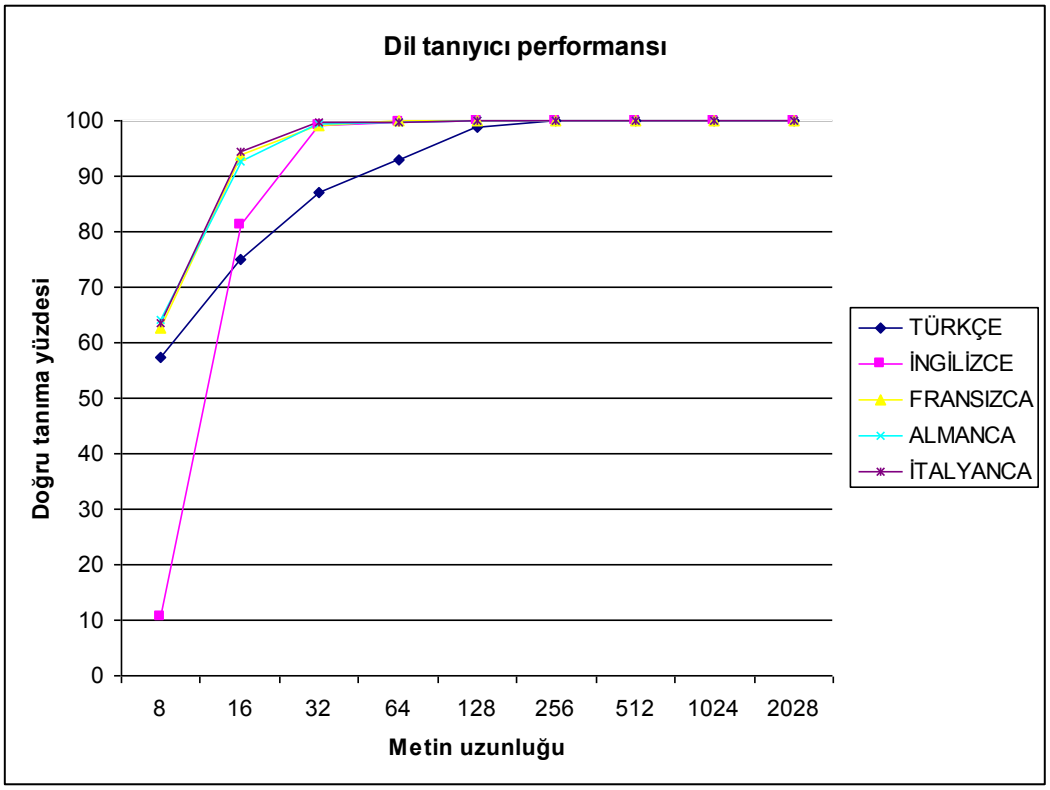
Çizelge 4.1'de görüldüğü gibi metin uzunluğu arttıkça tanıma yüzdesi yükselmekte ve Türkçe için 256, diğer diller için 128 karakter ve daha uzun metinler için doğru tanıma oranı yüzde yüze ulaşmaktadır.

Teste kullanılan bütün metinler UTF–8 olarak kodlanmıştır. Özellikle Türkçe gibi alfabesinde ANSI kod tablosu dışında karakter bulunduran diller için kodlamanın dilin doğru olarak tanınmasında son derece önemlidir.

Test sonuçları incelendiğinde Türkçe dışındaki dillerde doğru tanıma yüzdesi 32 karakterden itibaren 99% ve üzerinde olduğu görülmektedir. Türkçe için bu oran ancak 256 karakterden sonra yakalanabilmektedir.

Çizelge 4.1 Dil tanıma test sonuçları

UZUNLUK	TÜRKÇE	İNGİLİZCE	FRANSIZCA	ALMANCA	İTALYANCA
8	57,38	10,62	62,77	64,15	63,38
16	75,08	81,08	93,69	92,77	94,46
32	86,92	99,23	99,23	99,54	99,85
64	93,08	99,69	100	99,85	99,85
128	98,77	100	100	100	100
256	100	100	100	100	100
512	100	100	100	100	100
1024	100	100	100	100	100
2048	100	100	100	100	100



Şekil 4.1 Dil tanıma test sonuçları

Türkçe testi için kullanılan metinler incelendiğinde bazı metinlerin yabancı kelimler, yabancı özel isimler, Türkçe olmayan yemek, işletme, yerleşim birimi

isimleri bulunduğu, özellikle futbol ile ilgili metinlerde yabancı kelimelerin ağırlıklı olarak kullanıldığı görülmektedir.

Özellikle 256 harften kısa metinlerde Türkçe dil tanıma yüzdesinde diğer dillere göre düşük olmasının yukarıda sayılan nedenlerden kaynaklandığı değerlendirilmektedir.

4.2. Türkçe Kök Bulma ve Bilgi Erişim Test Sonuçları

Geliştirilen TurkishPrefixMatchFilter filtresi Milliyet koleksiyonu kullanılarak test edilmiştir [23]. Test iki aşamalı olarak gerçekleştirilmiştir.

Birinci aşamada Milliyet koleksiyonunda yer alan 408.314 haber önce Lucene kütüphanesinin `LetterTokenizer` sembolleştirici sınıfı kullanılarak sadece harflerden oluşan sembollere dönüştürülmüş sonrada semboller küçük harfe çevrilerek indekslenmiştir. Oluşturulan indeksin 875.202 eşsiz terim içerdiği tespit edilmiştir.

İkinci aşamada aynı koleksiyon yine `LetterTokenizer` sembolleştirici sınıfı kullanılarak sadece harflerden oluşan sembollere dönüştürülmüş ve küçük harfe çevrilmiştir. Bu aşamada ilave olarak semboller `TurkishPrefixMatchFilter` filtresinden geçirilerek kökleri bulunarak indekslenmiştir. Be kez oluşturulan indekste 193.262 eşsiz terim olduğu görülmüştür.

Yapılan test sonucunda elde edilen veriler karşılaştırıldığında `TurkishPrefixMatchFilter` filtresi kullanıldığı zaman terim sayısında 77.9% oranından bir azalma olduğu görülmektedir.

Genel bir fikir vermesi açısından "Hakem karşılaşma sırasında pozisyonlardaki takdirlerini çoğunlukla Hırvatlar lehine kullandı" cümlesinin her iki yöntemle analiz edilmesinden ortaya çıkan terimler Çizelge 4.2'de görüldüğü gibidir.

Yukarıda yapılan testlerden elde edilen sonuçlar tek başına anlamlı değildir. Amaç terim sayısının azaltılması değil bir bilgi erişimin sistemi olan arama motorumuzun etkinliğinin artırılmasıdır.

Testimizin ikinci aşamasında arama motorunun Türkçe bilgi erişim etkinliğini test etmek için Milliyet koleksiyonu kullanılmıştır [23]. Bu koleksiyon

408314 gazete haberi ve 72 bilgi ihtiyacı ve ilgililik değerlendirmelerinden oluşmaktadır.

Çizelge 4.2 Örnek bir cümlenin TurkishPrefixMatchFilter ile işlenmesi sonucu oluşan semboller

Sadece sembollere bölünmüş hali	**TurkishPrefixMatchFilter filtresinden geçirilmiş hali**
hakem	hakem
karşılaşma	karşılaş
sırasında	sıra
pozisyonlardaki	pozisyon
takdirlerini	takdir
çoğunlukla	çoğunluk
hırvatlar	hırvat
lehine	leh
kullandı	kullan

Koleksiyondaki bilgi ihtiyaçları tipik TREC [25] sorgularına benzer şekilde *başlık*, *tanım* ve *hikaye* olmak üzere üç bölümden oluşmaktadır. Bilgi erişim testleri *başlık* + *tanım* ve sadece *başlık* olmak üzere iki farklı şekilde gerçekleştirilmiştir.

Milliyet koleksiyonun ilgililik değerlendirmeleri tam değerlendirme olmadığından sistemimizin etkinliğini test etmek amacıyla "binary preference" *bpref* kullanılmıştır [26].

Kök bulma işlemi yapılmadan ve kök bulma işlemi yapılarak gerçekleştirilen *başlık* + *tanım* sorgularının bilgi erişim performansı Çizelge 4.3'de, sadece başlık kullanılarak gerçekleştirilen sorguların bilgi erişim performansı Çizelge 4.4'de görüldüğü gibidir.

Çizelge 4.3 Başlık ve Tanım sorgularının Bpref değerleri

	StandardAnalyzer	**TurkishPrefixMatchAnalyzer**
Bpref	0.3947	0. 5092
P5	0.6278	0.6889
P10	0.6042	0.6667
P15	0.5806	0.6556
P20	0.5688	0.6458
P30	0.5296	0.6097
P100	0.3676	0.4310
P200	0.2531	0.3020
P500	0.1339	0.1599
P1000	0.0760	0.0894

Çizelge 4.4 Sadece Başlık sorgularının Bpref değerleri

	StandardAnalyzer	**TurkishPrefixMatchAnalyzer**
Bpref	0.3506	0.4538
P5	0.5583	0.5639
P10	0.5250	0.5625
P15	0.4944	0.5444
P20	0.4708	0.5312
P30	0.4361	0.5014
P100	0.2917	0.3707
P200	0.2080	0.2661
P500	0.1112	0.1452
P1000	0.0659	0.0827

4.3. Toparlama Test Sonuçları

Arama motorunun çeşitli alt bileşenlerinin performansını test etmek mümkün olsa da özellikle toparlayıcı bileşeninin performansının test etmek için bir performans ölçütü mevcut değildir. Toparlayıcının performansı birçok faktöre dayandığından test edilmesi de son derece güçtür. Ancak testlerimiz sırasında

toparlayıcı beklediğimiz gibi herhangi bir çökme yaşamadan istikrarlı bir şekilde çalışmıştır. Çalışma zamanında oluşan hataların büyük kısmı HTML ve PDF metin çıkarıcısı gibi sınıfların içeriği işlerken oluşturdukları çalışma zamanı hatalarıdır.

Toparlayıcının performansını etkileyen birçok faktör bulunmaktadır. Bu faktörlerden en önemlileri şunlardır.

- Toparlayıcının kullanabileceği etkin bant genişliği
- Eldeki donanım kaynakları ve bu kaynakların performansları
- Sunuculara aşırı yüklenmekten kaçınmak için nezaket kurallarına uyulması gerekliliği

Toparlayıcının performansını etkileyen faktörlerden en önemlisi nezaket kurallarına uyulmasıdır. Elde yeterli bant genişliği olsa bile bir web sitesinden aynı anda sadece bir sayfa indirilememektedir. Ancak web sitelerinin alt alan adları ayrı birer web sitesi gibi kabul edilmektedir.

Aramam motorunu test etmek için basın konusu seçilmiş ve belirlenen web sitelerinin indekslenmesi amaçlanmıştır. Basın ile ilgili indekslenen web siteleri listesi Çizelge 4.9'dadır. Test amaçlı olarak kullanılan bilgisayarın özellikleri Çizelge 4.5'dedir.

Çizelge 4.5 Testte kullanılan bilgisayarın özellikleri

İşlemci	Intel Core Duo 2 Ghz
Önbellek	2 Mb
Hafıza	1 Gb DDR2 667 Mhz
Sabit Disk	120 Gb 5400 Rpm
Bant genişliği	2 Mbit ADSL

Basın ile ilgili sitelerin indekslenmesi 6 gün 13 saatte tamamlanmıştır. Toplam 5961463 bağlantı keşfedilmiş bunların 5956563 adedi başarıyla işlenmiştir. Test sonucunda tespit edilen HTTP cevap kodları ve oranları Çizelge 4.6'da görüldüğü gibidir. Çizelgede açıkça görülebileceği gibi kaynağın taşındığı

anlamına gelen HTTP 302 kodu ve kaynağın bulunamadığı anlamına gelen HTTP 404 hata kodu oranları oldukça yüksektir.

Çizelge 4.6 HTTP cevap kodlarının miktarı ve oranları

HTTP Cevap Kodu	**Miktarı**	**Oran (%)**
HTTP-200 OK	4847110	83,27
HTTP-302 Redirect-Found	489033	8,40
HTTP-404 Not Found	393985	6,76
HTTP-301 Moved Permanently	43355	0,74
HTTP-400 Bad Request	23899	0,41
HTTP-500 Internal Server Error	22363	0,38

Arama motorunun tespit ettiği ve indirdiği dokümanların MIME türü, miktarı ve oranları Çizelge 4.7'de görüldüğü gibidir.

Çizelge 4.7 İndirilen dokümanların MIME türleri, miktarları ve oranları

Dosya Tipi	**Miktarı**	**Oran (%)**
text/html	3119938	64,36
image/jpeg	912413	18,82
image/gif	197656	4,07
diğer	617103	12,73

Çizelge 4.7 incelendiğinde indirilen dosya tiplerini çok büyük bir bölümünü HTML web sayfaları ve JPEG formatındaki görüntü dosyaları oluşturmaktadır. Arama motoru aslında JPEG ve GIF gibi resim dosyalarını indirmemektedir. HTTP başlık bilgisi incelenerek eğer içerik türü bu formatlardan birisi ise indirme işlemi sonlandırılmaktadır.

İndirilen dokümanlardan 3046083 adedinden metin çıkarılarak indekse kaydedilmiştir. İndirilen ve indekslenen doküman sayıları arasındaki bu fark bazı dokümanlardan metin çıkarıldığında indekslenecek herhangi bir içeriğin tespit edilememesidir.

İndekslenen dokümanların dil tanıma sonuçları Çizelge 4.8'de görüldüğü gibidir. Türkçeden farklı dillerde tanınan dokümanlardan incelendiğinde sonuçların dil tanıma eklentisinin testleriyle paralellik gösterdiği görülmektedir. Dil tanıma eklentisinin performansının çok büyük oranda içeriğin dil kodlamasının doğru tanınmasına bağlı olduğu gözüne alındığında toparlayıcının içeriğin karakter kodlamasının tespitinde de yüksek bir isabet oranı yakaladığı değerlendirilmektedir.

Çizelge 4.8 Toparlayıcı dil tanıma sonuçları

Dil	**Miktar**	**Oran (%)**
Türkçe	3012754	98,90
Diğer	33329	1,10

Çizelge 4.9 Basın ile ilgili indekslenen web siteleri

No	**URL**	**No**	**URL**
1	http://www.aksam.com.tr/	11	http://www.turkiyegazetesi.com.tr/
2	http://www.cumhuriyet.com.tr/	12	http://www.gazetevatan.com/
3	http://www.dunyagazetesi.com.tr/	13	http://www.yenicaggazetesi.com.tr/
4	http://www.hurriyet.com.tr/	14	http://www.yenisafak.com.tr/
5	http://www.milligazete.com.tr/	15	http://www.zaman.com.tr/
6	http://www.milliyet.com.tr/	16	http://www.tercuman.com.tr/
7	http://www.radikal.com.tr/	17	http://www.haberturk.com/
8	http://www.sabah.com.tr/	18	http://www.aa.com.tr/
9	http://www.stargazete.com/	19	http://www.tercuman.com.tr/
10	http://www.takvim.com.tr/		

5. SONUÇLAR VE ÖNERİLER

Türkçe dokümanların etkin olarak indekslenmesi ve sorgulanmasına yönelik genişletilebilir yapıda bir dikey arama motoru geliştirilmiştir. Bu çalışmadan hareketle ilgi alanlarına yönelik özel ihtiyaçları karşılamak için daha detaylı analizler gerçekleştirecek, özel sıralama algoritmaları kullanan, daha detaylı sorgu ve analizlere imkan sağlayan bir dikey arama motorunun geliştirilmesi imkan dahilindedir.

Dikey arama motorunun temel unsurlarından birisi olan içeriğin ilgi duyulan alana göre etkin şekilde analizi edilmesi konusu ilgi alanında ileri seviyede bilgi sahibi olunmasına ihtiyaç göstermektedir. Ancak o zaman yapılacak analizler için temel teşkil edecek parametreler uygun şekilde tespit edilebilecektir.

Geliştirilen dikey arama motorunun daha etkin olarak kullanılabilmesi için ilgi alanının belirlenmesi ve bu alanla ilgili dokümanları işleyebilmek için içeriğin alanın özelliğine uygun bir şekilde analiz edilmesine ihtiyaç vardır. Medyayı takip etmek için kullanılan bir dikey arama motoruyla otomotiv sektörüne yönelik bir dikey arama motoru içeriğin işlenmesi, metnin sembolleştirilmesi ve köklerinin bulunmasında kullanılan yöntemler bakımından birbirinden farklı olacaktır.

Günümüzde birçok web sitesinin içeriği dinamik olarak üretilmektedir. Dinamik olarak üretilen bu sayfaların neredeyse tamamında menü ve başlık bilgileri aynen tekrarlanmakta, bir çok sayfada dinamik olarak üretilen reklam içeriği bulunmaktadır. Reklam içeriği bazen sayfanın yararlı içeriğinin birkaç katı kadar olabilmektedir. Özellikle haber sunan birçok web sitesinde en son eklenen başlıklar, en çok okunan haberler veya en çok yorumlananlar gibi bölümler bütün sayfaların içeriğine eklenmektedir. Bu durum saylardan çıkarılan asıl içeriğin yanında sorgulama sonuçlarını olumsuz yönde etkileyecek birçok bilginin de indekslenmesine neden olmakta ve sonuç olarak sorgu sonuçlarının ilgililik oranlarının azalmasına neden olmaktadır.

Son derece karmaşık bir sayfanın içerisinden yararlı içeriğin bulunup çıkarılmasının bilgi erişim etkinliğini artıracağı değerlendirilmektedir. Ancak bu tür bir analiz yapmak bizim konumuzun dışındadır.

Geliştirilen Türkçe kök bulma yöntemi sayesinde test sonuçlarında da açıkça gördüğümüz gibi Türkçe bilgi erişim performansının artmasına rağmen indekse kaydedilen eşsiz terim sayısında ciddi bir azalma gözlenmektedir. Terim sayısındaki bu azalmanın bilgi erişim kütüphanesinin performansının olumlu yönde etkileyeceği değerlendirilmektedir.

Dil tanıma sonuçlarından dokümanların dilini tanımak için ilk 256 karakterinin yeterli olduğun tespit edilmiştir. Dil tanımada ilk 256 karakterin kullanılmasının dil tanıma performansını artırdığı ve daha az kaynağa ihtiyaç duyduğu tespit edilmiştir.

Sonuç olarak dikey arama motorumuzun Türkçe dokümanlar için bir dikey arama motoru geliştirilmesi hedefine ulaşıldığı değerlendirilmektedir.

KAYNAKLAR

[1] Manning C.D., Raghavan P., Schütze H., *An Introduction to Information Retrieval*. Cambridge University Pres, England, 2008.

[2] Singhal A., *Modern Information Retrieval: A Brief Overview*, IEEE Data Engineering Bulletin 24(4), 2001.

[3] Anonim, *Text REtrieval Conference*, 2008. http://trec.nist.gov/overview.html

[4] Anonim, *Google 2.0: Google Universal Search*, 2008. http://searchengineland.com/070516-143312.php

[5] Barshinger, S., *The Emerging Opportunity in Vertical Search*, 2006. http://www.slackbarshinger.com/verticalsearch/

[6] Anonim, *Natural Language Processing*, 2008. http://en.wikipedia.org/wiki/Natural_language_processing

[7] Mohr G., *Introduction to Heritrix*, 2008. http://www.iwaw.net/04/Mohr.pdf

[8] Anonim, *Resmi Heritrix Web Sitesi*, 2008. http://crawler.archive.org/

[9] Anonim, *Arc File Format*, 2008. http://www.archive.org/web/researcher/ArcFileFormat.php

[10] Hull, D.A., *Stemming algorithms: A case study for detailed evaluation*, JASIS, 47(1), January, Special Issue on the Evaluation of Information Retrieval Systems, 1996.

[11] Sneath, P.H.A. and Sokal, R.R., *Numerical taxonomy: The principles and practice of numerical classification*, W.H. Freeman and Company, San Francisco, USA, 1973.

[12] Grefenstette, G., *Comparing two language identification schemes*, Proceedings of JADT 1995, 3rd International Conference on Statistical Analysis of Textual Data, 1995.

[13] Dunning T., *Statistical Identification of language*, Computing Research Laboratory, New Mexico State University: Technical Report, 94-273, 1994.

[14] Canvar, W.B. and Trenkle, J.M., *N-gram based Text Categorization*, Symposium on Document Analysis and Information Retrieval, University of Nevada, Las Vegas, 161-176, 1994.

[15] Kranig S., *Evaluation of Language Identification Methods*, Yüksek Lisans Tezi, University of Tübingen, International Studies in Computational Linguistics, Germany, 2005.

[16] Kikui G., *Identifying, the coding system and language, of on-line documents on the Internet*, In Proceedings of the 16th conference on Computational Linguistics - **2**, 652-657, 1996.

[17] Anonim, *Stemming*, 2008.
http://en.wikipedia.org/wiki/Stemming

[18] Dawson, J.L., *Suffix Removal for Word Conflation*, Bulletin of the Association for Literary and Linguistic Computing, **2**(3): 33-46, 1974.

[19] Lovins, J.B., *Development of a Stemming Algorithm*, Mechanical Translation and Computational Linguistics **11**, 22-31, 1968.

[20] Porter, M.F., *An Algorithm for Suffix Stripping*, Program, 14(3): 130-137, 1980.

[21] Porter, M.F., *Snowball*, 2008.
http://snowball.tartarus.org/index.php

[22] Gospodnetic, O., Hatcher, E., *Lucene In Action*, Manning Press, USA, 2005.

[23] Can F., Kocberberler, S., Balcik, E., Kaynak, C., Ocalan, H.C.,Vursavas, O.M., *First large-scale information retrieval experiments on Turkish texts*, SIGIR 2006, 627-628, 2006.

[24] Philipp K., *EuroParl: A Parallel Corpus for Statistical Machine Translation*, University of Southern California, Information Sciences Institute, USA, 2002.

[25] Alpha S., Dixon P., Liao C., and Yang C., *Oracle at TREC10* Notebook paper, 2001, http://trec.nist.gov/pubs/trec10/papers/orcltrec10.pdf

[26] Buckley, C. and Voorhees E.M, *Retrieval evaluation with incomplete information*, ACM SIGIR Conf, 2004.