SELÇUK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ

SUJEST, c.4, s.2, 2016
SUJEST, v.4, n.2, 2016
ISSN: 2147-9364 (Elektronik)

ELASTICSEARCH YÜK DENGELEME İŞLEMİNİN MANUEL YAPILANDIRILMASI ve BAŞARIM ÖLÇÜMÜ İÇİN YAZILIM GELİŞTİRİLMESİ

[1]Mustafa Ali AKCA, [2]Tuncay AYDOĞAN

[1]Süleyman Demirel Üniversitesi, Eğitim Fakültesi, Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü
[2]Süleyman Demirel Üniversitesi, Teknoloji Fakültesi, Yazılım Mühendisliği Bölümü
[1] mustafaakca@sdu.edu.tr, [2] tuncayaydogan@sdu.edu.tr

(Geliş/Received: 20.01.2016; Kabul/Accepted in Revised Form: 26.03.2016)

ÖZ: Geleneksel yöntemlerle işlenemeyen, saklanamayan devasa büyüklükteki bilgilerin oluşturduğu veri yığınına Big Data (Büyük Veri) denmektedir. Gün geçtikçe daha popüler hala gelen bu kavram, beraberinde bu verileri işleyebilecek araçların gereksinimini de ortaya çıkarmıştır. Bu büyük veri yığınının analizi ve saklanması için kullanılan araçlardan biri de Elasticsearchtür. Elasticsearch Java ile geliştirilmiş, açık kaynak kodlu Lucene tabanlı bir içerik analizi ve arama motorudur. Elasticsearch yapısı itibariyle dağıtık mimaride çalışabilen bir yazılımdır. Verileri aynı index içinde farklı shardlarda, aynı disk içinde farklı klasörlerde, aynı bilgisayar içerisinde farklı disklerde, aynı ağ içerisinde farklı sunucularda barındırabilir. Tüm bu seçenekler kullanıcıların ihtiyaçlarına göre şekillendirilebilir. Elasticsearch düğümü çalıştığı anda içinde bulunduğu kümede tüm indexler için aktif bir rol üstlenir. Diğer düğümlerle haberleşir ve yük dağılımı yapılmaya başlanır. Bu yük dağılımı normal şartlarda, düğüm üzerine düşen yükü parçalayarak performansı artırmayı hedeflemektedir. Ancak elasticsearch tarafından otomatik olarak yapılan bu yük dağılımı her zaman performans artırıcı etkiler oluşturmayabilir. Yapılan bu çalışmada geliştirilen yazılım ile Elasticsearch düğümleri üzerindeki yük takip edilebilmekte ve manuel yapılandırılmasına olanak sağlayabilmektedir. Kullanıcıların düğüm kaynaklarının kullanım oranlarını görebileceği, indexlere ait shardları manuel dağıtabileceği, otomatik shard tahsisini açıp kapatabileceği ve tüm bu yapılandırmaların verimliliğini test edebileceği bir yazılım geliştirilmiştir.

Anahtar Kelimeler: *Elasticsearch, Yük Dağılımı, Yük Dengeleme*

Software Development For Manual Configuration Of Elasticsearch Load Balancing And Performance Measurement

ABSTRACT: Huge amount of data stack which cannot be stored nor processed by traditional methods is called Big Data. This term, which is becoming more and more popular, led to the necessity of tools to process this data. One of the tools which is used for analysis and storage of this huge among of data is Elasticsearch. Elasticsearch is a content analysis and search server based on Lucene and developed in Java as open source. It is a software which can operate as distributed architectural structure. It also can store data in different shards in the same index, in different files in the same disk, in different disks in the same computer, and in different servers in the same network. All these options are shaped by the needs of users. As soon as Elasticsearch node starts working, it takes an active role in all indexes of clusters. It also connects with other nodes and the share of load takes place. This load distribution normally aims to increase performance by decreasing load in each node. However, this load distribution done

DOI: 10.15317/Scitech.2016218524

automatically by Elasticsearch might not always create effects which increase performance. With the software developed in this study, load in each Elasticsearch nodes are tracked and manual configuration is enabled. This software enables users to observe node activity rates, to distribute shards in indexes manually, to switch on and off shard automatically, to index all these configuration productivity, and to test as inquiry-based.

Key Words: *Elasticsearch, Load Distribution, Load Balancing*

GİRİŞ (INTRODUCTION)

Analizi, yönetimi ve işlenmesi geleneksel yöntem ve araçlarla mümkün olmayan büyük miktardaki verilere Big Data (Büyük Veri) denilmektedir (Ohlhorst, 2013). Önümüzdeki yıllarda birçok bilgisayarın bulut ağı üzerinde çalışacağı ve haberleşeceği, bu sebepten dolayı sürekli olarak büyük miktarda sayısal bilgiler üretileceği tahmin edilmektedir (Science Clouds, 2014). Günümüzde hergün devasa miktarlarda üretilen bu bilgilere bazı örnekler verilebilir;

- Youtube'a her dakika 48 saat uzunluğunda video yüklenmektedir (yılda 2,5 petabayt)
- Boing jet motorları 30 dakikalık uçuş için 10 terabayt veri üretebilmektedir.
- Square Kilometer Array (Kilometre Kara Dizgesi-SKA) isimli teleskobu saniyede 100 terabit veri oluşturabilmektedir (Hallaç, 2014)

Birçok farklı alanda üretilen bu bilgilerin yönetimi ve işlenmesi için geleneksel araçlara ek olarak bazı araçlar geliştirilmiş ve geliştirilmeye devam etmektedir. Elasticsearch bu araçlardan biridir. Elasticsearch Java ile geliştirilmiş, açık kaynak kodlu, Lucene tabanlı içerik arama ve analiz aracıdır. 2010 Yılında geliştirilmeye başlanmış Elasticsearch yerli/yabancı birçok kuruluş tarafından kullanılmaktadır. Google For Works kapsamında satın alınan Cloud hizmetlerinde sunuculara doğrudan kurulabilen, Google'ında bu bağlamda desteklemiş olduğu bir araçtır. Dağıtık mimari desteği sayesinde veriler aynı index içinde farklı shardlarda, aynı disk içinde farklı klasörlerde, aynı bilgisayar içerisinde farklı disklerde, aynı ağ içerisinde farklı sunucularda barındıralabilmektedir. Elasticsearchte varsayılan olarak her bir index 5 shard ve 1 replica ile oluşturulmaktadır. Yani her dosya kümemiz 5 parçaya bölünmekte ve her bir parçanın birer adet yedeği bulunmaktadır. Elasticsearch aynı zamanda kümeleme desteğiyle birden farklı sunucuda birbirleriyle haberleşen düğümler şeklinde çalışabilmektedir. Küme ismi aynı olan düğümler çalıştığı andan itibaren birbirlerini tanırlar ve aralarında yük dengeleme yapmaya başlarlar. Yük dengeleme (Load balancing), servislerin kesintisiz ve yedekli çalışabilmesi için birden fazla sunucunun tek bir servis için, tek bir sunucu gibi çalıştırılabilmesine olanak sağlayan teknolojidir (Netinternet, 2016). Elasticsearch yapısı gereği yük dengeleme işlemini otomatik olarak yapabilmektedir. Aynı sunucu içerisinde farklı disklerde çalışan düğümler ya da, farklı sunucularda çalışan düğümler aynı veri kümesinde çalışıyorlar ise düğümler aktif olduğu andan itibaren bu yük dengeleme işlemi başlamaktadır. Örneğin A düğümünde varsayılan olarak 5 shard ve replicasız oluşturulan bir index, aynı küme içinde B isimli yeni bir düğüm çalıştırıldığında parçalanarak bu iki düğüme dağıtılır. Bir düğümde aynı indexe ait 3 shard diğer düğümde ise 2 shard yer alacak şekilde yük dengeleme işlemi yapılmaktadır.

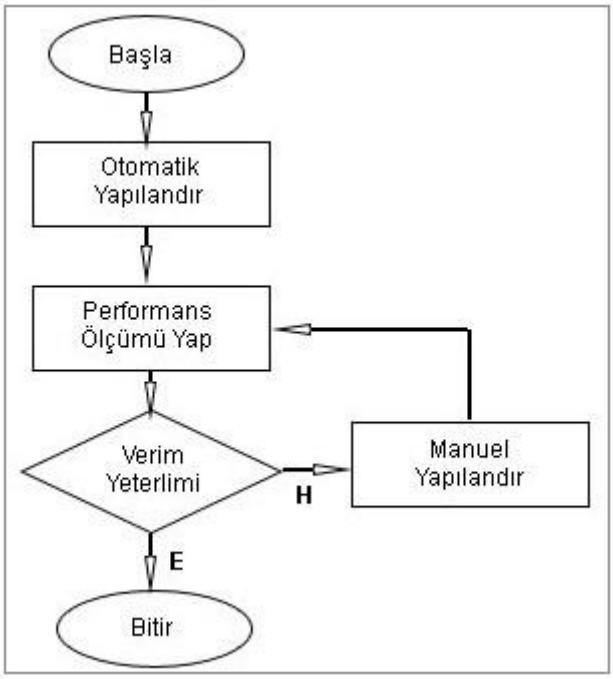
Ancak sunucu kaynaklarının kullanım oranlarına ve sorgulanan/analiz edilen verinin büyüklüğüne, sorgulama/analiz edilme sıklığına göre otomatik yapılan bu yük dengeleme işlemi her zaman performans artırıcı olumlu sonuçlar vermeyebilir. Elasticsearchte yer alan varsayılan optimizasyon işlemleri her uygulama için istenen sonuçlar ortaya çıkarmayabilir. Elasticsearch kümesinden en iyi performansı alabilmek için bir çok çalışma yapılmıştır (Chaudhary, 2014; Peschlow, 2014; Mikalauskas, 2015; Gheorghe, 2013).

Bu çalışmada Elasticsearchün otomatik yük dengeleme işlemi sonucunda performansı olumsuz etkileyebilecek durumlara, manuel yük dengeleme ile yapılandırma yapabilen bir yazılım ile çözüm

getirilmeye çalışılmıştır. Bu sayede ihtiyaçlar doğrultusunda tüm çalışmalara uyarlanabilecek, ve Elasticsearchün otomatik yük dengelemesinin verimsiz kaldığı durumlarda performansı artırımı sağlanılabilecektir. Geliştirilen yazılım HTML ve Javascript kullanılarak Elasticsearch ile RestFull API üzerinden haberleşebilmektedir. Bu sayede yazılım dilim bağımsız olarak tüm platformlarda çalışabilmektedir. Geliştirilen yazılım aynı küme içerisindeki tüm sunucu, düğüm ve düğümler üzerindeki indexleri takip edebilmektedir. Sunucu ve düğümlerin yük durumlarını izleyip, kullanıcıya manuel yapılandırma imkanı tanımakta ve performans testleri yapabilmesine olanak sağlamaktadır.

MATERYAL VE METOT (MATERIALS AND METHOD)

Geliştirilen yazılım Elasticsearch kümelerini izleme, shard dağılımlarını yapılandırma, indexleme ve sorgulama performans ölçümleri yapabilme temelleri üzerine inşa edilmiştir. Şekil 1'de akış şeması görülen yazılım, düğümler ilk çalıştığı anda shard dağılımlarını izler ve kullanıcıya gösterir. Kullanıcı Elasticsearch otomatik yapılandırması tamamlandıktan sonra indexleme ve sorgulama performans testleri yapar. Eğer verim yeterli değil ise otomatik yapılandırma kapatılır ve manuel yapılandırma işlemleri gerçekleştirilir. Sonuç en iyi hale gelene kadar performans ölçüm ve manuel yapılandırma işlemleri devam ettirilir.



Şekil 1. Yazılımın akış şeması
Figure 1. *The flow chart of software*

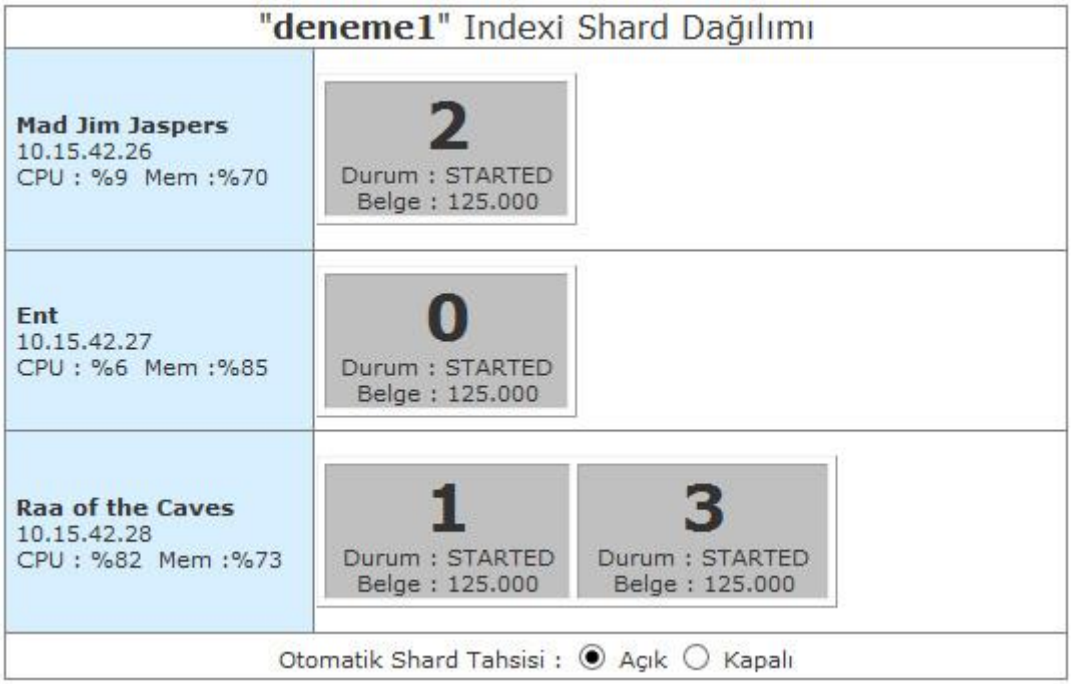
Yazılım dili bağımsız olarak tüm platformlarda web tarayıcı üzerinden çalışabilecek şekilde geliştirilen bu yazılım, küme üzerinde yer alan düğümleri RestFull API üzerinden yönlendiren HTML tabanlı olarak Ajax ve Restful API kullanılarak yapılmıştır.

Ajax teknolojisi (Asynchronous JavaScript and XML) ilk olarak 2005 yıllarında geliştirilmeye başlanılmıştır (Garret, 2005). Ajax yeni bir programlama dili olmayıp, etkileşimli uygulamalar geliştirmede kullanılan bir tekniktir (Vatansever ve Batık, 2011). Ajax tekniği ile dinamik sayfaların güncellenmesinde, sayfalarının tamamının yenilenmesi yerine sadece kısmi yenilemeler yapılarak sayfaların güncel tutulması hedeflenmiştir. Web sayfalarına kütüphane olarak eklenip kullanılabilir. Ajax kütüphanesi ile HTML sayfalar içerisinde POST ve GET işlemleri yapılabilir. Bu sayede HTML sayfalar sayfa yenilemesi yapılmadan güncellenebilir.

Restful, istemci ve sunucu arasında haberleşme ve veri iletişimi sağlayan bir mimaridir (İrgin, 2012). Bu iletişimde POST, PUT, DELETE, GET metodları kullanarak başta JSON olmak üzere bir çok farklı veri tipinde iletişim sağlayabilir.

Bu çalışmada Ajax kütüphanesi üzerinden Restful API kullanılarak küme üzerindeki düğümler HTML bir sayfa üzerinden takip edilmiş ve yapılandırılmıştır.

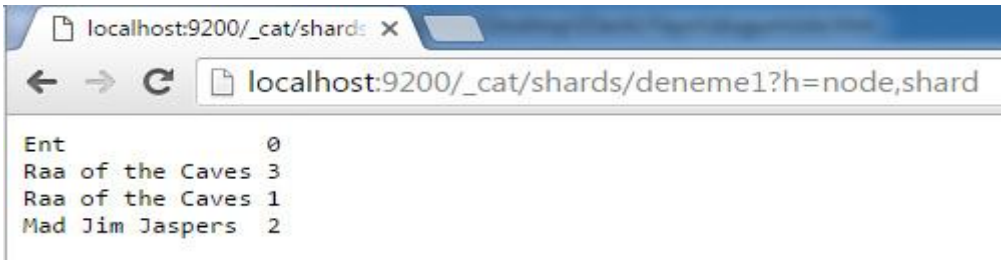
Elasticsearch dağıtık mimaride çalışabilen bir analiz ve arama aracıdır. Küme üzerinde birden fazla düğüm var ise oluşturulan indexleri otomatik olarak bu düğümlere dağıtır. Bu düğümler aynı bilgisayarda farklı disklerde çalışabileceği gibi aynı ağ üzerinde farklı bilgisayarlarda da çalışabilmektedir. Düğümler ilk çalıştırıldığı anda Elasticsearchün varsayılan konfigürasyonları gereği, düğüm, diğer düğümler üzerinden kendine yük paylaşımı yapar. Örneğin bir kümede çalışan bir index var ise ve bu index 4 shard'a sahip ise diğer düğüm çalıştığı zaman 2 shard otomatik olarak diğer düğüme taşınır. Elasticsearch bu işlemi düğüm üzerine binen yükü dağıtarak performansı artırmak amacıyla otomatik olarak yapmaktadır. Ancak teoride verimli gibi görünen bu işlem, uygulamada her zaman olumlu sonuçlar vermeyebilir. Yeni aktif edilen düğümün fiziksel özellikleri, devam etmekte olan diğer iş yükleri, ağ trafiğinin yoğunluğu gibi sebeplerden dolayı indexleme ve sorgulama hızları düşebilmektedir. Bu çalışmada geliştirilen yazılım Elasticsearch indexlerinin yük dağılım durumlarını anlık olarak takip edilebilmesine imkan tanıyan bir arayüze sahiptir.



Şekil 2. Index İzleme ve Manuel Yapılandırma Arayüzü

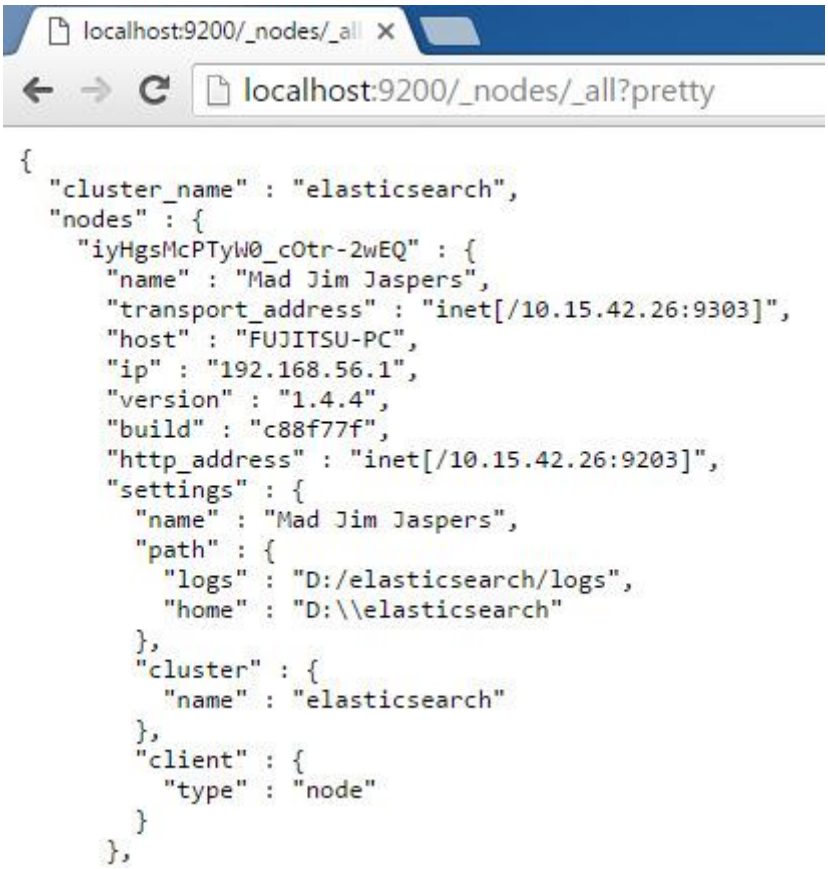
Figure 2. *Index Monitoring and Manual Configuration Interface*

Şekil 2'de, geliştirilen yazılımın Elasticsearch üzerinde yer alan indexlerin izlenebileceği bir arayüzü bulunmaktadır. Bu arayüzde de görüldüğü gibi "deneme1" isimli index 4 shard dosyasından oluşmaktadır. Bu shard dosyalarının 2 tanesi "Raa of the Caves" isimli düğümde yer alırken, diğer düğümlere birer tane shard tahsis edilmiştir. Indexte yer alan toplam belge sayısı 500.000 adet ve her bir düğümde 125.000 adet yer almaktadır. Elasticsearch Php My Admin ya da MS SQL Management Studio gibi bir takip ve yönetim ekranına sahip değildir. Şekil 1'de görülen tüm bilgiler Restful API üzerinden düğümden istenmektedir.



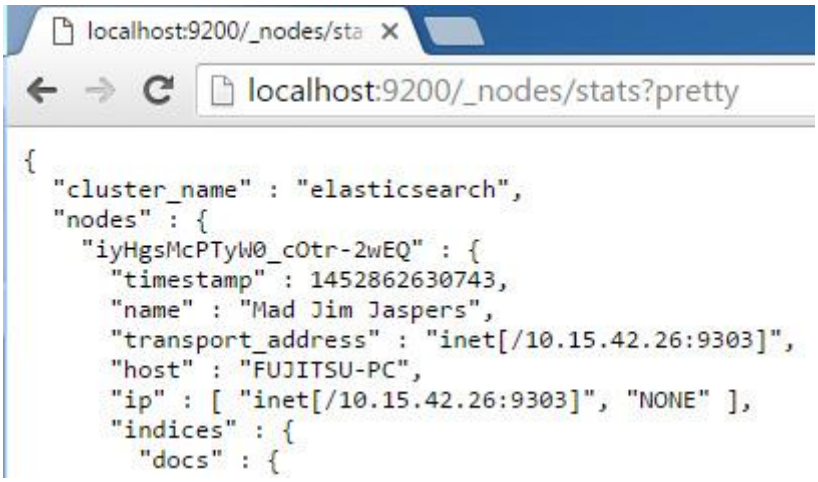
Şekil 3. Cat API'nin Tarayıcı Üzerinden Kullanılması
Figure 3. *Using the Cat API via a Browser*

Elasticsearch düğümlerin, indexlerin ve shardların izlenmesi için birçok farklı API desteği sunmaktadır. Bunlardan biri de Cat API'dir. Cat API ile Şekil 3'te görüldüğü gibi belirlenen indexe ait shardların şuan hangi düğüm üzerinde hizmet verdiği görülebilmektedir.



Şekil 4. Nodes API
Figure 4. *Nodes API*

Şekil 4'te görülen Nodes API sayesinde ise kümede kaç düğümün aktif olduğunu görülebilmektedir. Aynı zamanda o düğümlere ait bazı bilgiler de bulunmaktadır. Nodes API'den elde edilen bilgiler JSON formatında olup, HTML sayfa içerisine aktarılmadan önce bazı string işlemlerine tabi tutularak parçalanmaktadır.



Şekil 5. Nodes Stats API
Figure 5. *Nodes Stats API*

Şekil 5'te görülen Nodes Stats API ile düğümlere ait bazı bilgiler elde edilebilmektedir. Bu API düğümlerin işlemci kullanım yüzdelerini, ram kullanım yüzdelerini, aktif kaç adet belge barındırdıklarını gösterebilmektedir.

```
function shardtahsis() {
$.getJSON(baglanti + '/_nodes/_all/process?pretty', function(veri) {
        $.get(baglanti + "/_cat/shards/" + seciliindex, function(data) {
            $.getJSON(baglanti + '/_nodes/stats?pretty',
            function(istatistikveri) {
```

Şekil 6. Ajax ile 3 API'nin İç İçe Kullanımı
Figure 6. *Nested 3 API usage with Ajax*

Şekil 6'da görüşen Ajax kodlarında Şekil 2'de görülen ekran oluşturulurken Şekil 3, Şekil 4 ve Şekil 5'teki API'ler birlikte kullanılmıştır. Bu üç API'den gelen bilgiler string işlemleri ile ayrıştırıldıktan sonra düğüm, index ve shard bazlı olarak tekrar ilişkilendirilmiş ve Şekil 2'de görülen izleme ekranı oluşturulmuştur. Elasticsearch düğümlerinin, ilk çalıştıklarında varsayılan olarak otomatik shard tahsis özellikleri aktif durumdadır. Bu sebepten dolayı küme içinde başka bir düğüm aktif olduğunda otomatik yük dağılımı yapılabilmektedir. Geliştirilen bu arayüzde kullanıcı talebi doğrultusunda Şekil 7'de görüldüğü gibi otomatik shard tahsisi açılıp kapatılabilmektedir.

```
$.ajax({
    type: "PUT",
    async: false,
    url: baglanti + '/' + seciliindex + '/_settings',
    data: JSON.stringify({
        "index": {
            "routing": {
                "allocation": {
                    "disable_allocation": "false"
                }
            }
        }
    }),
```

Şekil 7. Otomatik Shard Tahsis Kontrolü
Figure 7. *Automatic Shard Allocation Control*

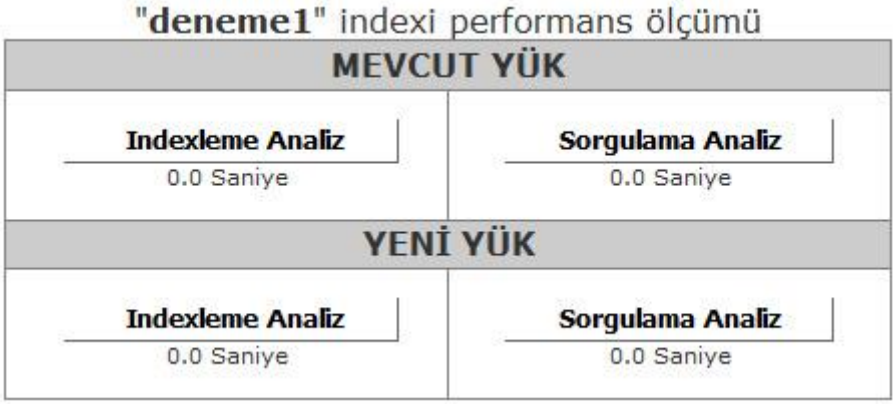
Kullanıcılar bu sayede yük dengelemelerini manuel olarak yapılandırabilmektedirler. Kullanıcılar düğümlerdeki yük dağılımını manuel yapmadan önce düğüm üzerindeki işlemci ve ram kullanımlarını kontrol etmektedirler. Kullanıcılar düğümleri inceleme neticesi sonucunda arayüzde görünen shard numaralarının üzerine tıklayarak sürükle-bırak ile Şekil 8'deki kodlar ile shardların düğüm lokasyonları değiştirilebilmektedir. Kullanıcılar bu sayede yük dengelemelerini manuel olarak yapılandırabilmektedirler

```
$.ajax({
    type: "POST",
    async: false,
    url: baglanti + '/_cluster/reroute',
    data: JSON.stringify({
        "commands": [{
            "move": {
                "index": seciliindex,
                "shard":
                document.getElementById(data).getAttribute("data-shard"),
                "from_node":
                document.getElementById(data).getAttribute("data-kaynakdugum"),
                "to_node": ev.target.id
            }
        }]
    }),
```

Şekil 8. Sürükle – Bırak İşlemi İle Shard Tahsisi
Figure 8. *Shard Allocation with Drag-Drop*

YAZILIMIN TEST EDİLMESİ (Software Testing)

Kullanıcıların Şekil 2'deki arayüzde düğüm ve index durumunu takip edip yapılandırdıktan sonra, bu işlemin verimliliğini test edebilmeleri için başka bir arayüz daha geliştirilmiştir.



Şekil 9. Performans ölçüm arayüzü
Figure 9. *Performance measurement interface*

Şekil 9'da görülen arayüzde kullanıcılar yapılandırmaları sonucu verimlilik ölçümü yapabilmektedirler. Bu verimlilik ölçümü 2 türde yapılabilmektedir. İlk ölçüm index üzerindeki mevcut yük ile sağlanmaktadır. Örneğin bu uygulamada test amaçlı oluştulan "deneme1" indexinde 500.000 kayıt yer almaktadır. Bu kayıt performans testleri için yeterli bir rakamdır. Kullanıcı 500.000 kayıtlı index üzerinden test yapmak istediğinde "Mevcut Yük" başlığı altındaki "Indexleme Analiz" ve "Sorgulama Analiz" butonlarını kullanarak sonuçları sayısal olarak görebilmektedir. Ancak yeni oluşturulacak ve kullanılacak bir indexte ileriye dönük olarak bir ölçüm düşünülüyor ise bu ölçümlerde "Yeni Yük" sekmesi altında yapılabilmektedir.

```
var start = Date.now();
$.get("http://localhost:9200/_plugin/basarim/index.test.file.txt",
function(datam) {
```

Şekil 10. Text dosyası içerisindeki 100.000 örnek kaydın alınması
Figure 10. *Taking 100,000 sample record in the text file*

```
$.ajax({
    type: "POST",
    async: false,
    url: 'http://localhost:9200/_bulk',
    data: datam,
    dataType: "json",
    contentType: "application/text; charset=utf-8",
    success: function(text) {
```

Şekil 11. Bulk API ile indexe toplu veri kaydedilmesi
Figure 11. *Saving bulk data to index with Bulk API*

Yeni yük sekmesinde oluşturulan indexe Bulk API ile Şekil 10'daki gibi text dosyası içerisinden 100.000 kayıt alınıp datam değişkenine yüklenmektedir. Daha sonra Şekil 11'deki Bulk API ile indexe bu veriler kaydedilmektedir. Tüm bu işlemler yapılırken işlem başlatılmadan önce bir timer başlatılıp, işlem bittikten sonra durdurularak arada geçen süreler ölçülmektedir.

Çizelge 1. Test sonuçları
Table 1. *Test Results*

Elasticsearch Otomatik Yapılandırma				Manuel Yapılandırma			Fark	
Düğümler	Shard Dağılım	Ortalama Süreler (100 Deneme)		Shard Dağılım	Ortalama Süreler (100 Deneme)		Başarım Artış	
		Indexleme	Sorgulama		Indexleme	Sorgulama	Indexleme	Sorgulama
Düğüm 1	2	6.03 sn	0.499 sn	2	4.74 sn	0.367 sn	~%27	~%35
Düğüm 2	0			1,3				
Düğüm 3	1,3			0				

Yazılım geliştirildikten sonra otomatik yapılandırma ve manuel yapılandırma arasındaki farkları ölçebilmek için bazı testler gerçekleştirilmiştir. Çizelge 1'de bu testlere ait bazı sonuçlar yer almaktadır. İlk olarak "deneme1" isminde 500.000 belge ve 4 sharda sahip olan bir index oluşturulmuş ve elasticsearchün bu indexi otomatik yapılandırması izlenmiştir. Elasticsearch Düğüm 1, Düğüm 2 ve Düğüm 3'e sırasıyla (2),(0),(1,3) shardlarını tahsis etmiştir. Yapılandırma işlemi bittikten sonra indexleme performansını ölçebilmek için 100.000 adet belge Bulk API ile indexlenmiştir. Bu işlem 100 defa tekrar edilmiş ve her bir işlemin ortalama 6.03 sn sürdüğü tespit edilmiştir. Indexleme işlemi sonrası 100 defa sorgulama işlemi yapılmış ve her bir işlemin ortalama 0.499 sn'lik süre tespit edilmiştir. Bu işlemler tamamlandıktan sonra sunucu fiziksel özellikleri, mevcut yoğunlukları göz önünde bulundurularak otomatik yapılandırma kapatılıp, manuel yapılandırma işlemi yapılmıştır. Düğüm 3'teki yük kendisinden daha iyi donanım imkanlarına sahip olan Düğüm 2'ye aktarılmış, Düğüm 2'deki yük de Düğüm 3'e aktarılmıştır. Manuel yapılandırma işlemleri bittikten sonra aynı testler tekrar yapılmış ve indexlemede 4.74 sn sorgulamada 0.367 sn lik süreler elde edilmiştir. Otomatik ve manuel yapılandırmadaki süreler oranlanacak olursa indexlemede yaklaşık %27 lik sorgulamada ise yaklaşık %35 lik verim artışı gözlemlenmiştir.

SONUÇ ve TARTIŞMALAR (RESULTS and DISCUSSIONS)

Bilişim teknolojilerinin kullanım alanlarının gelişmesiyle birlikte Büyük Veri (Big Data) kavramıda her geçen gün yaygınlaşmaya devam etmektedir. Sosyal medyada oluşturulan bilgiler, ticari sistemlerin oluşturdukları bilgiler, e-devlet uygulamalarının oluşturdukları veriler vb. birçok alanda, bir çok uygulama neticesinde veriler üretilmeye devam etmektedir. Devasa ölçüde oluşturulan bu verilerin standart saklama, arama ve analiz araçları ile yönetilmebilmesi oldukça zor bir iştir. Bunun için Büyük Veriler üzerinde bu işlemleri gerçekleştirebilecek birçok araç geliştirilmiştir. Bunlardan biri olan Elasticsearchün günümüzde kullanım oranı güngeçtikçe yaygınlaşmaktadır. Hem açık kaynak kodlu, hem de Lucene tabanı üzerine Java ile inşa edilmiş olması bu kullanımı artırmaya yardımcı olmaktadır.

Elasticsearch dağıtık mimaride çalışan bir araçtır. Bu sayede düğümler arası yük dengeleme işlemi yapabilir. Bu ayar Elasticsearchte varsayılan "aktif" olarak gelmektedir. Ancak kümede kullanılan düğümlerin fiziksel kaynakları kullanım oranları, ağ trafiği, sorgulama/indexleme çeşitlilikleri sebebiyle Elasticsearchün yapmış olduğu bu yük dengeleme işi her zaman verimli olmayabilir. Bu çalışmada Son kullanıcıya yönelik bir izleme ve yapılandırma arayüzü olmayan Elasticsearchte izleme ve yapılandırma işlemlerine olanak tanıyan bir yazılım geliştirilmiştir. Bu yazılım ile aynı zamanda otomatik yük dengelemesi pasifleştirilerek kullanıcının daha verimli bir yapılandırma oluşturulabilmesine imkan tanıyan bir arayüz tasarlanmıştır. Bu yazılımın, yazılım dili bağımsız olarak tüm platformlarda çalışabilmesi için HTML, Ajax ve Restful teknolojileri kullanılmıştır. Kullanıcılar bu yazılım sayesinde kendi fiziksel kaynakları, sorgulama-indexleme işlemlerindeki yoğunluk farklılıkları, ağ kaynaklarına göre manuel yapılandırmalar yapabilmekte, aynı zamanda bu yapılandırmaların verimliliğini ölçebilmektedir. Geliştirilen arayüzler son kullanıcıya yönelik yapılmış olup takip edilmesi ve

kullanılması oldukça basittir. Geliştirilen uygulama çeşitli fiziksel imkanlara sahip Elasticsearch kümelerinde başarıyla test edilmiştir. Kullanıcılar bu uygulama ile ihtiyaçlarına en uygun yapılandırmaları kendileri oluşturabilecekler ve verimlilik testi yapabileceklerdir. Geliştirilen bu uygulamanın aynı zamanda, gelecekte "Otomatik ve Akıllı" yapılandırmalar yapabilecek çalışmalara da ışık tutacağı düşünülmektedir

KAYNAKLAR (REFERENCES)

Chaudhary, M., "9 Tips on ElasticSearch Configuration for High Performance", https://www.loggly.com/blog/nine-tips-configuring-elasticsearch-for-high-performance/, Son Erişim : 25.03.2016

Garrett, J.J., "Ajax: A New Approach to Web Applications", http://adaptivepath.org/ideas/ajax-new-approach-web-applications/, Son Erişim : 14.12.2015

Gheorghe, R., "Elasticsearch Refresh Interval vs Indexing Performance", https://sematext.com/blog/2013/07/08/elasticsearch-refresh-interval-vs-indexing-performance/, Son Erişim : 25.03.2016

Hallaç, İ.R., "Büyük veri analizlerinde dağıtık makine öğrenmesi algoritmalarının kullanılması", Yüksek Lisans Tezi, Fırat Üniversitesi, 2014

İrgin, D., "REST ve RESTful Web Servis Kavramı", http://www.denizirgin.com/post/2012/05/28/REST-RESTful-Web-Service.aspx/, Son Erişim : 14.12.2015

Mikalauskas, A., "Quick Way To Improve Elastıcsearch Performance On A Single Machine", http://www.speedemy.com/quick-way-to-improve-elastic-search-performance-on-a-single-machine/, Son Erişim : 25.03.2016

Netinternet, https://www.netinternet.com.tr/yuk-dengeleme, Son Erişim: 04.01.2016

Ohhorst, F., "Turning Big Data Into Big Money", Big Data Analytics, , New Jersey, AB.D., 2013

Peschlow, P., "Elasticsearch Indexing Performans Cheatsheet", https://blog.codecentric.de/en/2014/05/elasticsearch-indexing-performance-cheatsheet/,
Son Erişim : 25.03.2016

Science Clouds., https://portal.futuregrid.org/., Son Erişim : 03.07.2014

Vatansever, F., Batık, Z., "İnternette Ajax Tekniği", 6th International Advanced Technologies Symposium, Elazığ, Türkiye, 2011