SDU International Journal of Technological Sciences
Vol. 10, No 3, December 2018
pp. 10-19

Computational Technologies

METİN VERİLERİNİN HADOOP MAP/REDUCE VE KLASİK TEKNİKLER İLE İŞLENMESİ ARASINDA BİR PERFORMANS KARŞILAŞTIRILMASI

Mustafa KAYA *, Tuncay AYDOĞAN

Geliş Tarihi/ Received: 14.11.2018, Kabul tarihi/Accepted: 25.12.2018

Özet

Günümüzde teknolojinin hızlı gelişimine paralel olarak elde edilen muazzam sayıdaki bilgilerin işlenmesi büyük önem arz etmektedir. Bu "Büyük Veri"'lerin analizi için farklı yazılım araçları geliştirilmiştir. Büyük verileri işlemede kullanılan araçlardan biri olan Hadoop, yüksek maliyetli bilgisayarlar yerine sıradan bilgisayarlardan oluşan küme (cluster) üzerinde büyük verileri işleyen bir sistem olması sebebiyle tercih edilmekte ve kullanımı yaygınlaşmaktadır. Bu çalışmada, kelime sayma amacıyla Hadoop Map/Reduce tekniği kullanılarak geliştirilen bir uygulama ile Java dilinde geliştirilen diğer uygulamanın çalışma performansları işlem süresi açısından ortaya konulmuştur. Hadoop kullanılarak geliştirilen uygulamanın 1.000.000 adet kelimeyi Java uygulamasına göre 481 kat daha hızlı işlediği görülmüştür.

Anahtar Kelimeler: Hadoop, Map/Reduce, Veri analizi.

A PERFORMANCE COMPARISON BETWEEN HADOOP MAP / REDUCE AND CLASSIC TECHNIQUES OF TEXT DATA

Abstract

Nowadays, it is very important to process the huge amount of information obtained in parallel with the rapid development of technology. Different software tools are developed for analysis of these "Big Data". Hadoop, is one of the systems used in big data processing, is a system that processes big data on a cluster consisting of ordinary computers instead of high cost computers and its usage is widespread. In this study, the working performance of the other application developed in Java language with an application developed using Hadoop Map / Reduce technique for word counting has been demonstrated in terms of processing time. It has been seen that Hadoop application processes 1.000.000 words 481 times faster than Java application.

Key Words: Hadoop, Map/Reduce, data analysis.

*Bilgisayar Teknolojileri, Bafra Meslek yüksekokulu, Ondokuz Mayıs Üniversitesi, Samsun, Türkiye
E-posta: mustafa.kaya@omu.edu.tr

1. Giriş

Bilgi, verinin işlenmiş halidir. Karmaşık ve büyük boyutlu bir veriden çıkarılan anlamlı içerik bilgi olarak adlandırılır. Günümüzde teknolojinin hızlı gelişimine paralel olarak elde edilen veri miktarları da gün geçtikçe büyümektedir. Sadece Google günde 1,5 trilyon sayfayı analiz etmektedir (Oracle, 2016). CISCO tarafından yayınlanan, Görsel Ağ Endeksi'nin (Visual Networking Index-VNI) 2014- 2019 Küresel Mobil Veri Trafiği Öngörü Raporuna göre 2021 yılına kadar dünya çapında mobil veri trafiği yılda 1.000 Eksabayta (EB) ulaşacaktır (Cisco 2017). Bu büyük verileri kısa sürede işleyebilmek için farklı yazılım araçları geliştirilmiştir.

Hadoop, yüksek maliyetli bilgisayarlar yerine sıradan bilgisayarlardan oluşan kümeler (cluster) üzerinde büyük verileri işleyen bir sistemdir. Bu sistem, Hadoop Distributed File System (HDFS) olarak isimlendirilen dağıtılmış büyük veri dosyalarını Map ve Reduce teknikleri/süreçleri ile modelleyerek işleyen, Java programlama diliyle geliştirilmiş açık kaynaklı bir kütüphanedir. Bu kütüphane Dean ve Ghemawat çalışmasında HDFS'nin oldukça düşük hata toleransıyla, yüksek verimlilik ve güvenilirlikle sunucular üzerinde işlem yapmak için tasarlandığını, Map/Reduce'un ise Google tarafından büyük veri grupları için önerilen bir programlama modeli olduğunu belirtmişlerdir (Padhy, P. R. 2013).

Alkan, yaptığı çalışmasında belli bir ağ içinde önceden belirlenen eşik değerde kendisini tekrar eden küçük ağları bulmaya çalışmıştır. Bu çalışma için açık kaynak kodlu bir ortam olan Hadoop tercih edilmiş ve yöntemin büyük ağlara ölçeklenebildiği ve veri miktarının artmasına paralel olarak yöntemin daha yüksek verimle çalıştığı belirlenmiştir (Alkan, S. 2010).

Er, Hadoop ortamında genetik algoritmanın paralelleştirilmesini çalışmıştır. Problem olarak "Gezgin Satıcı Problemi" seçilmiştir. Ortaya çıkan sonuçlar daha önceki uygulamalarda elde edilen sonuçlarla karşılaştırılmış, Map/Reduce programlama modeli ile daha hızlı ve daha iyi sonuçlar elde edildiği gözlemlenmiştir (Er, H. R. 2013).

Bir başka çalışmada ise Çetin, Hadoop üzerinde kullanılan Hive, Map/Reduce ve Pig teknolojileriyle dağıtık RDFS çıkarsama işlemi gerçekleştirip, performanslarını gözlemlemiştir. Ayrıca doküman indeksleme yöntemi de kullanılmıştır. Bunun amacı performansları yükseltmektir. Yapılan çalışmalar sonunda Map/Reduce tekniğinin performans açısından daha iyi olduğu görülmüştür. Ayrıca dokuman indeksleme yöntemi Map/Reduce'un çalışma performansını daha da arttırmıştır (Çetin, Y. 2014).

Alewiwi ise, şifrelenmiş metin verilerini çözmek için yapılan karşılaştırma sayısını büyük ölçüde azaltan yeni bir teknik geliştirmiştir. Yapılan çalışma için HDFS ve Map/Reduce programlama modelinden faydalanılmıştır. Çalışma sonunda yöntemin diğer tekniklere göre çok daha hızlı olduğunu gözlemlemiştir (Alewiwi, M. 2015).

Yılmaz, büyük verilerin analizini yüksek maliyetli ve büyük sistemli yapılar yerine düşük maliyetli ve çok daha az yer kaplayan ARM mimarili mikro bilgisayarlarla nasıl yapılabileceğini belirtmiştir. Çalışmasını Hadoop kullanarak gerçekleştirmiştir (Yılmaz, R. A. 2015).

Çetinkaya, yaptığı çalışmasında k-means algoritmasının Hadoop Map/Reduce metoduna uyarlanması ve etkin analizlerin yapılabilirliği incelenmiştir. Yapılan incelemelerde k-means algoritmasının uyarlanması başarılı bir şekilde gerçekleştirilmiştir. Ayrıca elde edilen

sonuçların analizi neticesinde mevcut müşteri gruplarının oluşturulma dinamikleri yerine k-means ile müşteri gruplarının oluşturulmasının karlılığı artıracağı ve daha net sonuçlar üreteceği gözlemlenmiştir (Çetinkaya, S., 2016).

(Gurbuz S., Aydın G., 2017) bilimsel makalelerin sınıflandırılmasında büyük veri araçlarının özellikleri incelemiş, Hadoop üzerinde yapılan çalışmanın makine sayısı/işlem süresi performansları sunulmuştur.

Ulusal ve uluslararası literatürde çeşitli büyük veri türleri ile Hadoop, başka büyük veri işleme araçları ve geleneksel algoritmik çözümler arasında performans karşılaştırmasının yapıldığı başka yayına da rastlanmaktadır.

Bu çalışmada ise detayları ileri bölümlerde verilen 25.000 ile 1.000.000 adet arasında değişen sayıda kelime içeren altı farklı .txt dosyası ile veri seti oluşturulmuş, söz konusu dosyaların içerisinde kelimelerin kaç defa tekrar edildiğinin analizi yapılarak Hadoop ve klasik yöntemin performans farkı araştırılmıştır. Bunun için hem Hadoop üzerinde Java dili kullanılarak hem de NetBeans editöründe Java dili ile iki ayrı program geliştirilmiştir. Hadoop Map/Reduce tekniği Java programlama dili ile geliştirildiğinden performansı karşılaştırılacak program için de Java tercih edilmiştir.

2. Hadoop Map/Reduce Tekniği

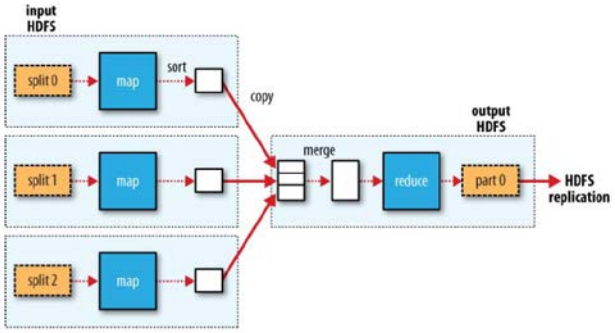
Hadoop, Map/Reduce tekniğini kullanarak büyük veri dosyalarının paralel şekilde işlenmesini sağlayan açık kaynak kodlu bir yapıdır (Lam, C. 2010). Bilgisayar kümeleri üzerinde çalışan dağıtık programlar için alt yapı desteği sunmaktadır (Er, H. R. 2013).

Hadoop, kullanıcı etkileşimine ihtiyaç duymadan verilerin ilgili düğüme taşınmasını ve olası bir donanımsal sorunda kurtarma işlemlerinin otomatik olarak gerçekleştirilmesini sağlar. Kullanılan Map/Reduce tekniğinde, yazılan program bilgisayar kümesinde bulunan bir düğümde (node) çalışabilecek küçük parçalara ayrılır.

Map/Reduce büyük veri kümelerindeki verilere en hızlı şekilde erişip işlemeyi sağlayan bir programlama modelidir (Dean, J. J. and Ghemawat, S. 2008). İlk olarak Jeffrey Dean ve Sanjay Ghemawat tarafından ortaya atılan bu model, Google sunucularında kayıtlı verilerin paralel işlenmesi amacıyla geliştirilmiştir. Günlük olarak kaydedilen web sayfalarının kütüğe yazılması sonucu oluşan veri kümesi içerisinde çok kısa sürede sonuç üretecek şekilde arama yapmak için geliştirilmiştir. Binlerce paralel çalışan bilgisayarı kullanabilecek şekilde tasarlanmıştır. Bu noktada dağıtık sistemlerdeki verinin parçalanması, dağıtılması ve işlemin paralelleştirilmesi gibi problemler ortaya çıkmıştır. Bu problemleri aşabilmek için yük dengeleme (load balancing), verinin dağıtılması ve hata toleransı işlemleri için bir soyutlama modeli tasarlanmıştır. Map/Reduce tekniği bu nedenlerle kullanıcı tarafından kolayca kullanılabilir ve dağıtık sistem üzerinde kolayca işlem yapmaya izin veren bir ara yüze sahiptir. Kullanıcı, sadece map ve reduce fonksiyonlarını yazarak paralelleştirilmesini istediği işi tanımlar, sistem bu iş için gerekli veri bölütlenmesini, paralelleştirmeyi ve hata yönetimini kendi yönetir. Map/Reduce kullanarak paralelleştirilebilen farklı uygulamalar mevcuttur. Bunlara örnek olarak web sayfası erişim frekansı tarama ve bulma, arama motorlarında kelime veya dosya arama (Ghemawat, S., vd. 2003), büyük boyutlu astronomik imgeler üzerinde işlem yapmak (Wiley, K., vd. 2011), dağıtık olarak sıralama algoritmaları çalıştırmak ve paralel görüntü işlemek (Krishna, M., vd. 2010) gibi paralel hesaplama gerektiren bazı işlemler verilebilir. Map/Reduce modeli daha sonra Apache yazılım topluluğu

tarafından Hadoop adında açık kaynak kodlu bir proje olarak yeniden geliştirilmiştir. Hadoop ücretsiz bir sistem olduğundan çok geniş bir geliştirici topluluğu tarafından kullanılmaktadır. Hadoop kullanıcıları arasında Yahoo gibi büyük kurumlar da mevcuttur.

Şekil 1.'deki veri akış diyagramında görülen Hadoop'ta paralelleştirilecek hesaplama bir görev (job) olarak tanımlanır. Bu görev, geliştirici tarafından kodlanan map ve reduce fonksiyonlarını içerir. Bu soyutlamada ilham kaynağı Lisp gibi fonksiyonel dillerdeki benzer fonksiyonlar olmuştur. Map/Reduce tekniği bu fonksiyonların birçok kopyasını paralel olarak farklı makinelerdeki işleyicilerde çalıştırır. Map fonksiyonu anahtar/değer (key/value) ikilisinden oluşan bir kayıt bilgisini girdi olarak alır ve yine anahtar/değer ikilisinden oluşan bir çıktı verisini oluşturur. Çok sayıda paralel çalışan bu Map fonksiyonları çıktılarını ara çıktı (intermediate output) olarak tutarlar ve bu çıktılar reduce fonksiyonlarına girdi olarak gönderilir. Tanımlı reduce fonksiyonlarını çalıştıran işleyiciler bu ara çıktıları girdi olarak kabul edip bunları tekrardan düzenleyip birleştirerek nihai çıktıyı oluşturur. Bu yeniden düzenleme sırasında aynı anahtar değerine sahip olan ara çıktıların değerleri birleştirilir.



Şekil 1. Map/Reduce veri akış diyagramı (White, T. 2009)

Map/Reduce görevini, mimarisini oluşturan önemli kavramlar ile gerçekleştirir;

Dağıtık sistemi oluşturan fiziksel bilgisayarlardan her birine Makine denir. Her makine de çalışan paralelleşmiş her bir Map/Reduce iş parçacığı İşleyici (Task) olarak tanımlanır. Map işleyicisi ve Reduce işleyicisi olarak çeşitleri vardır. Düğüm (Node), Hadoop tarafından her bir makineye verilen dağıtık sistemdeki rol veya tanımdır ve her makine aslında bir düğüm olarak görev almaktadır. İşleyicilerin dosya sisteminden okudukları ve işleyecekleri veri parçacığına Girdi Parçacığı (Input Split) denir. Map fonksiyonları tarafında oluşturulan ara çıktı verisi de Ara Çıktı (Intermadiate Output) dır. Bu veri anahtar/değer ikilisi şeklinde olup Reduce fonksiyonuna girdi olarak verilir.

Map/Reduce görevi aşağıdaki adımlarla özetlenebilir:

- Hadoop Map/Reduce tekniğinde kayıtlı olan büyük boyutlu dosyalar varsayılan 64 MB boyutunda bloklara ayrılır.
- Map işleyicisi girdi parçacığı verisini sistemden okuyarak bundan Map fonksiyonu için anahtar/değer ikilisini içeren kayıt verilerini oluşturur. Kayıt girdileri ile kullanıcı tarafından oluşturulmuş olan Map fonksiyonları çağrılır. Map fonksiyonlarının oluşturduğu ara çıktı ilgili makine tarafından Reduce fonksiyonuna gönderilmek üzere geçici hafızaya ve sonra da yerel diske kaydedilir. Reduce fonksiyonuna gönderildikten sonra bu veri silinir.
- Oluşturulan ara çıktı verileri Reduce işleyicilerine paylaştırılır ve Reduce işleyicileri bu verileri okuyarak ara anahtar verilerine göre sıralar ve guruplar. Böylelikle aynı

anahtar değerine sahip kayıtların değerleri gruplandırılmış olur. Kullanıcı isterse bu gruplama ve sıralama şeklini değiştirebilir.

- Reduce fonksiyonu içerisine gruplandırılmış yeni anahtar/değer ikilileri girdi olarak verilir. Kullanıcı tarafından oluşturulmuş reduce fonksiyonu nihai işleme tabi tutup oluşturduğu çıktıları çıktı dosyasına ekler.
- Yönetici düğüm, görevi çalıştıran kullanıcıya görevin tamamlanma bilgilerini sunar ve görevi sonlandırır (Demir, İ. 2012).

3. Hadoop Performans Değerlendirmesi

Geliştirilen programların performanslarını denemek için, aralarında birer boşluk karakteri bulunan 5 kelimelik satırlardan oluşan 25.000, 50.000, 125.000, 250.000, 500.000 ve 1.000.000 adet kelimelik veri setleri içeren altı farklı .txt dosyası oluşturuldu.

Programlar Intel Core 2 Duo T8300 2.40 GHZ işlemci, 4 GB DDR2 RAM kapasitesine sahip, Windows Vista 32 bit işletim sistemi bulunan bilgisayarda çalıştırıldı.

3.1. Java Uygulaması

Hadoop uygulaması ile performansı karşılaştırılan, Şekil 2'de görülen Java uygulaması NetBeans editöründe Java dili ile kodlanmıştır.

```
package siliversonra;
import java.io.BufferedReader;
import java.io.FileReader;
import javax.swing.JOptionPane;

public class kelime_hesaplama
{
  public static void main(String[] args)
  {
    String []kelime;
    String cumle="";
    long baslangic = System.nanoTime();
    try
    {
     FileReader fr=new FileReader("deneme_25bin.txt");
     BufferedReader br=new BufferedReader(fr);
     String str;

     while((str=br.readLine())!=null)
    {        cumle=cumle+str;      }

     kelime=cumle.split(" ");
     int sayac;
      for(int i=0; i<kelime.length; i++)
      {sayac=0;
       for(int m=0; m<i; m++)
       {     if(kelime[i].equals(kelime[m]))
                {     i=i+1;     }
        }

      for(int k=0; k<kelime.length; k++)
      {      if(kelime[i].equals(kelime[k]))
                sayac++;
       }
      System.out.println(kelime[i]+"  "+sayac);
      }

     br.close();
}
  catch (Exception hata)
        {         }
   long bitis = System.nanoTime();
   long calisma_suresi = bitis - baslangic;
   double saniye = (double)calisma_suresi/1000000000;
   System.out.println(saniye);
}
}
```

Şekil 2. Performansı karşılaştırılan Java uygulaması

Şekil 2'deki programın algoritmik adımları "deneme_25bin.txt" dosyasının okuma modunda açılması ile başlar. Dosyadaki veriler satırlar biçiminde okunmakta, tüm satırlar cümle değişkeninde string veri olarak birleştirilmektedir. Cümle değişkenindeki string veri java string fonksiyonları kullanılarak " " (boşluk) karakterine göre kelimelerine ayrıştırılarak her kelimenin kaç tane olduğu sayılır. Programın başlangıcına ve bitişine geçen süreyi hesaplamak için başlangıç ve bitiş anındaki zaman bilgisini alan kodlar yazılmıştır. İki zaman arasındaki fark verilerin işlenme süresi olan programın çalışma süresini vermiştir. Bu program tüm veri seti dosyaları için ayrı ayrı çalıştırılmıştır.

Her bir .txt dosyasındaki verileri işlemek için geçen süre Tablo 1'de görülmektedir. Tablodan görüldüğü gibi dosyadaki kelime sayısı katlanarak arttıkça işlem süresin de katlanarak artış göstermektedir.

Tablo 1. Veri seti dosyalarının Java uygulaması ile işlenme süreleri

Dosyada Veri Adedi	**Geçen Süre (saniye)**
25.000 kelimelik .txt dosyası	1,38
50.000 kelimelik .txt dosyası	4,26
125.000 kelimelik .txt dosyası	33,17
250.000 kelimelik .txt dosyası	158,17
500.000 kelimelik .txt dosyası	930,63
1.000.000 kelimelik .txt dosyası	3363,18

3.2. Hadoop Uygulaması

Java uygulamasının performansı ile karşılaştırılacak diğer uygulama Hadoop üzerinde Map/Reduce tekniği ile tek bilgisayar üzerinde yani Standalone Mode ortamında geliştirilmiştir. Geliştirilen uygulamanın Map ve Reduce fonksiyon kodları Şekil 3'de gösterilmiştir.

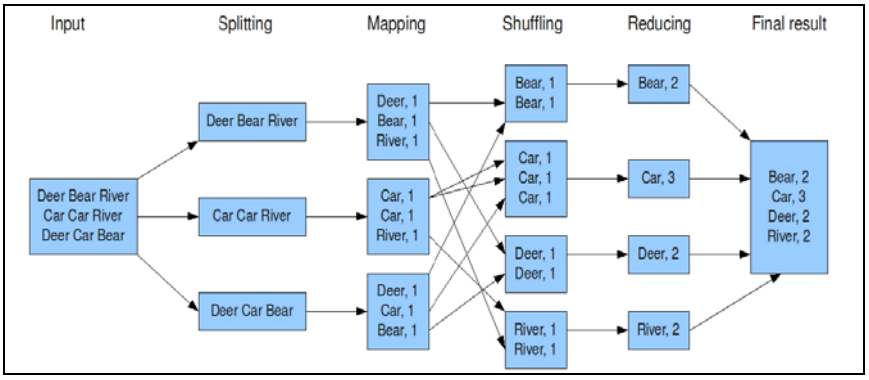
```
import java.io.IOException;
import java.util.StringTokenizer;
import javax.swing.JOptionPane;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount
{
static long baslangic = System.nanoTime();

public static class TokenizerMapper
extends Mapper<LongWritable, Text, Text, IntWritable>
{
private final static IntWritable one = new IntWritable(1);
private Text kelime = new Text();

public void map(LongWritable key, Text value, Context
context) throws IOException, InterruptedException
 {
  StringTokenizer itr = new
StringTokenizer(value.toString());
      while (itr.hasMoreTokens())
      {     word.set(itr.nextToken().toUpperCase());
        context.write(kelime, one);
       }
  }
}
```

```
public static class IntSumReducer
extends Reducer<Text,IntWritable,Text,IntWritable>
{
 private IntWritable sonuc = new IntWritable();

public void reduce(Text key, Iterable<IntWritable>
values,Context context) throws IOException,
InterruptedException
 {
   int toplam = 0;
   for (IntWritable val : values)
   {     toplam =toplam+ val.get();    }
    sonuc.set(toplam);
    context.write(key, sonuc);
   }
}

public static void main(String[] args) throws Exception
{
long bitis = System.nanoTime();
long sure = bitis - baslangic;
double saniye = (double)sure/1000000000;
System.out.println(saniye);

 Configuration conf = new Configuration();
 Job job = Job.getInstance(conf, "word count");
 job.setJarByClass(WordCount.class);
 job.setMapperClass(TokenizerMapper.class);
 job.setCombinerClass(IntSumReducer.class);
 job.setReducerClass(IntSumReducer.class);
 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(IntWritable.class);
 FileInputFormat.addInputPath(job, new Path(args[0]));
 FileOutputFormat.setOutputPath(job, new Path(args[1]));
 System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Şekil 3. Dosyadan okunan verileri saymak için geliştirilen Hadoop ortamında yazılan Java kodları

Şekil 3'deki programın map fonksiyonu, Hadoop kütüphanelerindeki string fonksiyonlarını kullanarak dosyadaki kelimeleri gruplandırıyor. Reduce fonksiyonu ise map fonksiyonundan gelen kelime grupları birleştirilerek her bir kelimenin kaç defa geçtiği hesaplanır. Program başına ve sonuna veri işleme süresini hesaplamak amacıyla gerekli kodlar eklenmiştir.

Map/Reduce fonksiyonları ile dosyadaki her kelimenin kaç tane olduğunu hesaplayan programının çalışma süreçleri kavramsal olarak Şekil 4'de gösterilmiştir. Program hesaplamayı Input, Splitting, Mapping, Shuffling, Reducing ve Final Result aşamaları ile tamamlamaktadır.



Şekil 4. Hadoop Map/Reduce tekniği ile geliştirilen programın kavramsal çalışma şeması (Kumar, A. 2018)

Input aşaması, işlenecek verilerin bulunduğu bir .txt dosyasından yapılan okuma işlemidir. Splitting aşamasında veriler kullanıcının isteğine göre belirlenecek büyüklüklerdeki bloklara ayrılır. Kullanıcı herhangi bir tanımlama yapmazsa veriler varsayılan olarak 64 MB'lık bloklara ayrılır. Mapping aşamada dosyada bulunan bütün kelimeler key(word)/value(1) şeklinde bölümlere ayrılır. Shuffling aşamasında, Map fonksiyonu çalıştıktan sonra elde edilen veriler aynı anahtar değerli veriler ile bir araya getirilerek aynı kelime grupları oluşturulur ve Reduce fonksiyonuna yönlendirilir. Reducing aşamasında Mapping aşamasından elde edilen çıktılar üzerinden toplama işlemi uygulanarak kelime sayıları bulunur. Final Result aşaması, Reducing aşamasında elde edilen kelime sayılarının bir .txt sonuç dosyasına yazdırılmasıdır.

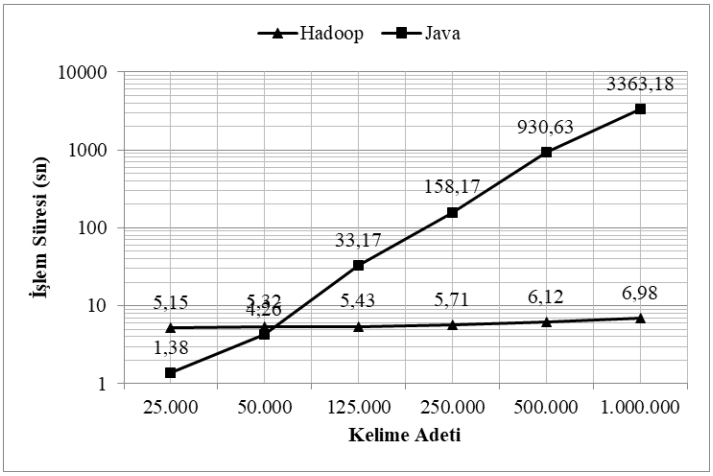
Her bir .txt dosyasındaki verileri işlemek için geçen süre Tablo 2'de görülmektedir. Tablodan görüldüğü gibi dosyadaki kelime sayısı katlanarak arttıkça işlem süresi az artış göstermektedir.

Tablo 2. Veri seti dosyalarının Hadoop uygulaması ile işlenme süreleri

Dosyada Veri Adedi	**Geçen Süre (saniye)**
25.000 kelimelik .txt dosyası	5,15
50.000 kelimelik .txt dosyası	5,32
125.000 kelimelik .txt dosyası	5,43
250.000 kelimelik .txt dosyası	5,71
500.000 kelimelik .txt dosyası	6,12
1.000.000 kelimelik .txt dosyası	6,98

4. Sonuçlar

Yapıları 3.Bölümde anlatılan uygulamaların altı farklı veri seti dosyası ile çalışması sonucu elde edilen işlem süresi performasları Şekil 5'deki grafik üzerinde gösterilmiştir.



Şekil 5. Hadoop ve Java uygulamalarının kelime sayısı/işlem süresi performans grafiği

Şekil 5 incelediğinde kelime sayısı 25.000 ve 50.000 olan dosyalarda Java uygulaması işlem süresi 1,38sn ve 4,26sn gerçekleşerek Hadoop uygulamasına göre daha hızlı görünse de, kelime sayısı arttıkça Java uygulamasının işlem süresinin arttığı görülmektedir.

Java uygulamasında 125.000 adet kelime sayısı 33,17sn'de işlenirken 250.000 adet kelime için süre yaklaşık 4,7 katına çıkıp 158,17sn olmaktadır. 500.000 adet kelime için bu süre yaklaşık 930,63sn (15,51 dakika) olmaktadır. 1.000.000 adet kelime için ise işlem süresi 3363,18sn (56 dakika) gerekmektedir.

Hadoop uygulamasındaki işlem süreleri kelime sayısı arttıkça çok fazla değişmemektedir. Hadoop 25.000 adet kelimeyi 5,15sn ve 50.000 adet kelimeyi 5,32sn sürelerde işlemiştir. Bu süreler Java uygulamasına oranla 3,7 ve 1,2 kat daha yavaştır. Ancak kelime sayısı arttıkça performans artışı daha net görülmektedir. 125.000 adet kelime için 5,43sn, 250.000 adet kelime için işlem süresi 5,71sn, 500.000 adet kelime sayısı için 6,12sn ve son olarak 1.000.000 adet kelime için ise 6,98sn süren hesaplama işlemi Java uygulamasından 481 kat daha hızlı gerçekleşmiştir. Bu fark büyüklükleri grafikte ancak logaritmik olarak ifade edilebilmiştir. Bu durum Hadoop Map/Reduce tekniğinin büyük verileri işlemek için geleneksel yöntemlerden çok daha verimli olduğunu göstermektedir.

Kaynaklar

Alewiwi, M. 2015. Efficient And Secure Document Similarity Search Over Cloud Utilizing Mapreduce. Sabancı Üniversitesi, Mühendislik Ve Doğa Bilimleri Enstitüsü, Doktora Tezi, 133s, İstanbul, 2015.

Alkan, S. 2010. A Distributed Graph Mining Framework Based On Mapreduce, Middle East Technical University, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 47s, Ankara, 2010.

Cisco 2017. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper, 2016–2021, https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html, 24.12.2018.

Çetin, Y. 2014. Mapreduce Kullanarak RDFS Üzerinde Dağıtık Çıkarsama. TOBB Ekonomi ve Teknoloji Üniversitesi, Fen Bilimleri Enstitüsü, 68s, Ankara, 2014.

Çetinkaya, S. 2016. Hadoop/Mapreduce Teknolojisi Kullanılarak Hızlı Tüketim Sektöründe Büyük Veri Analizi. İstanbul Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 82s, İstanbul, 2016.

Dean, J. J. and Ghemawat, S. 2008. "MapReduce: Simplified data processing on large clusters," Communications of the ACM, 2008, 51, 107–113, 2008.

Demir, İ. 2012. Hadoop Tabablı Büyük Ölçekli Görüntü İşleme Altyapısı. Kocaeli Üniversitesi Fen Bilimleri Enstitüsü Yüksek Lisans Tezi, 87s, Kocaeli, 2012.

Er, H. R. 2013. Gezgin Satıcı Probleminin Hadoop Üzerinde Çalışan Paralel Genetik Algoritma İle Çözümü. Kocaeli Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 83s, Kocaeli, 2013.

Ghemawat, S., Gobioff, H., Leung, S. 2003. The Google File System, Proc. of the 19th ACM Symp. on Operating System Principles, 29–43, 2003.

Gurbuz S., Aydın G., 2017. Büyük Veri Teknolojileri ile Bilimsel Makalelerin Sınıflandırılması, (UBMK'17) 2nd International Conference on Computer Science and Engineering.

Krishna, M., Kannan, B., Ramani, A., Sathish, S. J. 2010. Implementation and Performance Evaluation of a Hybrid Distributed System for Storing and Processing Images from the Web. Cloudcom, 2010 IEEE Second International Conference on Cloud Computing Technology and Science, 762-767, 2010.

Kumar, A. 2018. Basics Of Big Data Analytics Hadoop, https://www.slideshare.net/AmbujKumar4/basics-of-big-data-analytics-hadoop, 30.04.2018.

Lam, C. 2010. Hadoop in Action. Manning Publications Co., 301p, Stamford,USA, 2010.

Oracle, 2016. An Enterprise Architect's Guide to Big Data, 49p.

Padhy,P. R. 2013. Big Data Processing With Hadoop-MapReduce In Cloud Systems. International Journal of Cloud Computing and Services Science, 2(1), 16-27, 2013.

Yılmaz, R. A. 2015. Düşük Maliyet İle Mikro Süper Bilgisayar Oluşturma Ve Apache Hadoop Entegrasyonu. Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 106s, İstanbul, 2015.

White, T. 2009. Hadoop: The Definitive Guide. O'Reilly Media, Inc., 625p, Sebastopol, USA, 2009.

Wiley, K., Connolly, A., Krugho, S., Gardner, J., Balazinska, M., Howe, B., Kwon, Y. Y. and Bu Y, 2011. Astronomical Image Processing with Hadoop, 2011.